Express Mail Label No. ER203763807US

Query Engine For Processing Voice Based Queries Including Semantic Decoding

RELATED APPLICATIONS

The present application claims priority to and is a continuation-in-part of all of the following applications:

1) Serial No. 09/439,145 entitled Distributed Real Time Speech Recognition System, attorney docket no. PHO 99-001;
2) Serial No. 09/439,173 entitled Speech Based Learning/Training System, attorney docket no. PHO 99-002;
3) Serial No. 09/439,174 entitled Internet Server with Speech Support for Enhanced Interactivity – attorney docket no. PHO 99-003;
4) Serial No. 09/439,060 entitled Intelligent Query Engine For Processing Voice Based Queries – attorney docket no. PHO 99-004, now U.S. Patent No. 6,615,172;

The above are hereby incorporated by reference herein.

FIELD OF THE INVENTION

The invention relates to a system and an interactive method for rapidly and accurately processing speech queries through the use of statistical processing and semantic decoding. The system is particularly applicable to INTERNET based applications for e-learning, e-commerce, e-support, search engines and the like, so that a user can intelligently engage in a real-time question/answer session that emulates a human dialog experience.

BACKGROUND OF THE INVENTION

The INTERNET, and in particular, the World-Wide Web (WWW), is growing in popularity and usage for both commercial and recreational purposes, and this trend is expected to continue. This phenomenon is being driven, in part, by the increasing and widespread use of personal computer systems and the availability of low cost INTERNET access.

The emergence of inexpensive INTERNET access devices and high speed access techniques such as ADSL, cable modems, satellite modems, and the like, are expected to further accelerate the

mass usage of the WWW.

Accordingly, it is expected that the number of entities offering services, products, etc., over the WWW will increase dramatically over the coming years. Until now, however, the INTERNET "experience" for users has been limited mostly to non-voice based input/output devices, such as keyboards, intelligent electronic pads, mice, trackballs, printers, monitors, etc. This presents somewhat of a bottleneck for interacting over the WWW for a variety of reasons.

First, there is the issue of familiarity. Many kinds of applications lend themselves much more naturally and fluently to a voice-based environment. For instance, most people shopping for audio recordings are very comfortable with asking a live sales clerk in a record store for information on titles by a particular author, where they can be found in the store, etc. While it is often possible to browse and search on one's own to locate items of interest, it is usually easier and more efficient to get some form of human assistance first, and, with few exceptions, this request for assistance is presented in the form of a oral query. In addition, many persons cannot or will not, because of physical or psychological barriers, use any of the aforementioned conventional I/O devices. For example, many older persons cannot easily read the text presented on WWW pages, or understand the layout/hierarchy of menus, or manipulate a mouse to make finely coordinated movements to indicate their selections. Many others are intimidated by the look and complexity of computer systems, WWW pages, etc., and therefore do not attempt to use online services for this reason as well.

Thus, applications which can mimic normal human interactions are likely to be preferred by potential on-line shoppers and persons looking for information over the WWW. It is also expected that the use of voice-based systems will increase the universe of persons willing to engage in e-commerce, e-learning, etc. To date, however, there are very few systems, if any, which permit this type of interaction, and, if they do, it is very limited. For example, various commercial programs sold by IBM (VIAVOICE™) and Kurzweil (DRAGON™) permit some user control of the interface (opening, closing files) and searching (by using previously trained URLs) but they do not present a flexible solution that can be used by a number of users across multiple cultures and without time consuming voice training. Typical prior efforts to implement voice based functionality in an INTERNET context can be seen in U.S. Patent no. 5,819,220 incorporated by reference herein.

Another issue presented by the lack of voice-based systems is efficiency. Many companies are now offering technical support over the INTERNET, and some even offer live operator

assistance for such queries. While this is very advantageous (for the reasons mentioned above) it is also extremely costly and inefficient, because a real person must be employed to handle such queries. This presents a practical limit that results in long wait times for responses or high labor overheads. An example of this approach can be seen U.S. Patent no. 5,802,526 also incorporated by reference herein. In general, a service presented over the WWW is far more desirable if it is "scalable," or, in other words, able to handle an increasing amount of user traffic with little if any perceived delay or troubles by a prospective user.

In a similar context, while remote learning has become an increasingly popular option for many students, it is practically impossible for an instructor to be able to field questions from more than one person at a time. Even then, such interaction usually takes place for only a limited period of time because of other instructor time constraints. To date, however, there is no practical way for students to continue a human-like question and answer type dialog after the learning session is over, or without the presence of the instructor to personally address such queries.

Conversely, another aspect of emulating a human-like dialog involves the use of oral feedback. In other words, many persons prefer to receive answers and information in audible form. While a form of this functionality is used by some websites to communicate information to visitors, it is not performed in a real-time, interactive question-answer dialog fashion so its effectiveness and usefulness is limited.

Yet another area that could benefit from speech-based interaction involves so-called "search" engines used by INTERNET users to locate information of interest at web sites, such as the those available at *YAHOO®.com*, *METACRAWLER®.com*, *EXCITE®.com*, etc. These tools permit the user to form a search query using either combinations of keywords or metacategories to search through a web page database containing text indices associated with one or more distinct web pages. After processing the user's request, therefore, the search engine returns a number of hits which correspond, generally, to URL pointers and text excerpts from the web pages that represent the closest match made by such search engine for the particular user query based on the search processing logic used by search engine. The structure and operation of such prior art search engines, including the mechanism by which they build the web page database, and parse the search query, are well known in the art. To date, applicant is unaware of any such search engine that can easily and reliably search and retrieve information based on speech input from a user.

There are a number of reasons why the above environments (e-commerce, e-support, remote learning, INTERNET searching, etc.) do not utilize speech-based interfaces, despite the

many benefits that would otherwise flow from such capability. First, there is obviously a requirement that the output of the speech recognizer be as accurate as possible. One of the more reliable approaches to speech recognition used at this time is based on the Hidden Markov Model (HMM) – a model used to mathematically describe any time series. A conventional usage of this technique is disclosed, for example, in U.S. Patent No. 4,587,670 incorporated by reference herein. Because speech is considered to have an underlying sequence of one or more symbols, the HMM models corresponding to each symbol are trained on vectors from the speech waveforms. The Hidden Markov Model is a finite set of ***states***, each of which is associated with a (generally multi-dimensional) probability distribution. Transitions among the states are governed by a set of probabilities called ***transition probabilities.*** In a particular state an outcome or ***observation*** can be generated, according to the associated probability distribution. This finite state machine changes state once every time unit, and each time t such that a state j is entered, a spectral parameter vector O_t is generated with probability density $B_j(O_t)$. It is only the outcome, not the state visible to an external observer and therefore states are "hidden" to the outside; hence the name Hidden Markov Model. The basic theory of HMMs was published in a series of classic papers by Baum and his colleagues in the late 1960's and early 1970's. HMMs were first used in speech applications by Baker at Carnegie Mellon, by Jelenik and colleagues at IBM in the late 1970's and by Steve Young and colleagues at Cambridge University, UK in the 1990's. Some typical papers and texts are as follows:

1. L.E. Baum, T. Petrie, "Statistical inference for probabilistic functions for finite state Markov chains", Ann. Math. Stat., 37: 1554-1563, 1966
2. L.E. Baum, "An inequality and associated maximation technique in statistical estimation for probabilistic functions of Markov processes", Inequalities 3: 1-8, 1972
3. J.H. Baker, "The dragon system – An Overview", IEEE Trans. on ASSP Proc., ASSP-23(1): 24-29, Feb. 1975
4. F. Jeninek et al, "Continuous Speech Recognition: Statistical methods" in Handbook of Statistics, II, P.R. Kristnaiad, Ed. Amsterdam, The Netherlands, North-Holland, 1982
5. L.R. Bahl, F. Jeninek, R.L. Mercer, "A maximum likelihood approach to continuous speech recognition", IEEE Trans. Pattern Anal. Mach. Intell., PAMI-5: 179-190, 1983
6. J.D. Ferguson, "Hidden Markov Analysis: An Introduction", in Hidden Markov Models for Speech, Institute of Defense Analyses, Princeton, NJ. 1980.

7. H.R. Rabiner and B.H. Juang, "Fundamentals of Speech Recognition", Prentice Hall, 1993

8. H.R. Rabiner, "Digital Processing of Speech Signals", Prentice Hall, 1978

More recently research has progressed in extending HMM and combining HMMs with neural networks to speech recognition applications at various laboratories. The following is a representative paper:

9. Nelson Morgan, Hervé Bourlard, Steve Renals, Michael Cohen and Horacio Franco (1993), Hybrid Neural Network/Hidden Markov Model Systems for Continuous Speech Recognition. *Journal of Pattern Recognition and Artificial Intelligence*, Vol. 7, No. 4 pp. 899-916. Also in I. Guyon and P. Wang editors, *Advances in Pattern Recognition Systems using Neural Networks*, Vol. 7 of a Series in Machine Perception and Artificial Intelligence. World Scientific, Feb. 1994.

All of the above are hereby incorporated by reference. While the HMM-based speech recognition yields very good results, contemporary variations of this technique cannot guarantee a word accuracy requirement of 100% exactly and consistently, as will be required for WWW applications for all possible all user and environment conditions. Thus, although speech recognition technology has been available for several years, and has improved significantly, the technical requirements have placed severe restrictions on the specifications for the speech recognition accuracy that is required for an application that combines speech recognition and natural language processing to work satisfactorily.

In contrast to word recognition, Natural language processing (NLP) is concerned with the parsing, understanding and indexing of transcribed utterances and larger linguistic units. Because spontaneous speech contains many surface phenomena such as disfluencies, - hesitations, repairs and restarts, discourse markers such as '*well*' and other elements which cannot be handled by the typical speech recognizer, it is the problem and the source of the large gap that separates speech recognition and natural language processing technologies. Except for silence between utterances, another problem is the absence of any marked punctuation available for segmenting the speech input into meaningful units such as utterances. For optimal NLP performance, these types of phenomena should be annotated at its input. However, most continuous speech recognition systems produce only a raw sequence of words. Examples of conventional systems using NLP are shown in U.S. Patent Nos. 4,991,094, 5,068,789, 5,146,405 and 5,680,628, all of which are incorporated by reference herein.

Second, most of the very reliable voice recognition systems are speaker-dependent, requiring that the interface be "trained" with the user's voice, which takes a lot of time, and is thus very undesirable from the perspective of a WWW environment, where a user may interact only a few times with a particular website. Furthermore, speaker-dependent systems usually require a large user dictionary (one for each unique user) which reduces the speed of recognition. This makes it much harder to implement a real-time dialog interface with satisfactory response capability (i.e., something that mirrors normal conversation – on the order of 3 – 5 seconds is probably ideal). At present, the typical shrink-wrapped speech recognition application software include offerings from IBM (VIAVOICE™) and Dragon Systems (DRAGON™). While most of these applications are adequate for dictation and other transcribing applications, they are woefully inadequate for applications such as NLQS where the word error rate must be close to 0%. In addition these offerings require long training times and are typically are non client-server configurations. Other types of trained systems are discussed in U.S. Patent No. 5,231,670 assigned to Kurzweil, and which is also incorporated by reference herein.

Another significant problem faced in a distributed voice-based system is a lack of uniformity/control in the speech recognition process. In a typical stand-alone implementation of a speech recognition system, the entire SR engine runs on a single client. A well-known system of this type is depicted in U.S. Patent No. 4,991,217 incorporated by reference herein. These clients can take numerous forms (desktop PC, laptop PC, PDA, etc.) having varying speech signal processing and communications capability. Thus, from the server side perspective, it is not easy to assure uniform treatment of all users accessing a voice-enabled web page, since such users may have significantly disparate word recognition and error rate performances. While a prior art reference to Gould et al. – U.S. Patent No. 5,915,236 - discusses generally the notion of tailoring a recognition process to a set of available computational resources, it does not address or attempt to solve the issue of how to optimize resources in a distributed environment such as a client-server model. Again, to enable such voice-based technologies on a wide-spread scale it is far more preferable to have a system that harmonizes and accounts for discrepancies in individual systems so that even the thinnest client is supportable, and so that all users are able to interact in a satisfactory manner with the remote server running the e-commerce, e-support and/or remote learning application.

Two references that refer to a distributed approach for speech recognition include US Patents 5,956,683 and 5,960,399 incorporated by reference herein. In the first of these, US Patent 5,956,683 – Distributed Voice Recognition System (assigned to Qualcomm) an implementation of a

distributed voice recognition system between a telephony-based handset and a remote station is described. In this implementation, all of the word recognition operations seem to take place at the handset. This is done since the patent describes the benefits that result from locating of the system for acoustic feature extraction at the portable or cellular phone in order to limit degradation of the acoustic features due to quantization distortion resulting from the narrow bandwidth telephony channel. This reference therefore does not address the issue of how to ensure adequate performance for a very thin client platform. Moreover, it is difficult to determine, how, if at all, the system can perform real-time word recognition, and there is no meaningful description of how to integrate the system with a natural language processor.

The second of these references - US Patent 5,960,399 – Client/Server Speech Processor/Recognizer (assigned to GTE) describes the implementation of a HMM-based distributed speech recognition system. This reference is not instructive in many respects, however, including how to optimize acoustic feature extraction for a variety of client platforms, such as by performing a partial word recognition process where appropriate. Most importantly, there is only a description of a primitive server-based recognizer that only recognizes the user's speech and simply returns certain keywords such as the user's name and travel destination to fill out a dedicated form on the user's machine. Also, the streaming of the acoustic parameters does not appear to be implemented in real-time as it can only take place after silence is detected. Finally, while the reference mentions the possible use of natural language processing (column 9) there is no explanation of how such function might be implemented in a real-time fashion to provide an interactive feel for the user.

Companies such as Nuance Communications and Speech Works which up till now are the leading vendors that supply speech and natural language processing products to the airlines and travel reservations market, rely mainly on statistical and shallow semantics to understand the meaning of what the users says. Their successful strategy is based on the fact that this shallow semantic analysis will work quite well in the specific markets they target. Also to their advantage, these markets require only a limited amount to language understanding.

For future and broader applications such as customer relationship management or intelligent tutoring systems, a much deeper understanding of language is required. This understanding will come from the application of deep semantic analysis. Research using deep semantic techniques is today a very active field at such centers as Xerox Palo Alto Research Center (PARC), IBM, Microsoft and at universities such as Univ. of Pittsburg [Litman, 2002], Memphis [Graesser, 2000], Harvard [Grosz, 1993] and many others.

In a typical language understanding system there is typically a parser that precedes the semantic unit. Although the parser can build a hierarchical structure that spans a single sentence, parsers are seldom used to build up the hierarchical structure of the utterances or text that spans multiple sentences. The syntactic markings that guide parsing inside a sentence is either weak or absent in a typical discourse. So for a dialog-based system that expects to have smooth conversational features, the emphasis of the semantic decoder is not only on building deeper meaning structures from the shallow analyses constructed by the parser, but also on integrating the meanings of the multiple sentences that constitute the dialog.

Up till now there are two major research paths taken in deep semantic understanding of language: informational and intentional. In the informational approach, the focus is on the meaning that comes from the semantic relationships between the utterance-level propositions (e.g. effect, cause, condition) whereas with the intentional approach, the focus is on recognizing the intentions of the speaker (e.g. inform, request, propose).

Work following the informational approach focuses on the question of how the correct inferences are drawn during comprehension given the input utterances and background knowledge. The earliest work tried to draw all possible inferences [Reiger, 1974; Schank, 1975; Sperber & Wilson, 1986] and in response to the problem of combinatorial explosion in doing so, later work examined ways to constrain the reasoning [DeJong, 1977; Schank et al., 1980; Hobbs, 1980]. In parallel with this work, the notions of conversational implicatures (Grice, 1989) and accomodation [Lewis, 1979] were introduced. Both are related to inferences that are needed to make a discourse coherent or acceptable. These parallel lines of research converged into abductive approaches to discourse interpretation [e.g., Appelt & Pollack,1990; Charniak, 1986; Hobbs et al., 1993; McRoy & Hirst, 1991; Lascarides & Asher, 1991; Lascarides & Oberlander, 1992; Rayner & Alshawi, 1992]. The informational approach is central to work in text interpretation.

The intentional approach draws from work on the relationship between utterances and their meaning [Grice, 1969] and work on speech act theory [Searle, 1969] and generally employs artificial intelligence planning tools. The early work considered only individual plans [e.g., Power, 1974; Perrault & Allen, 1980; Hobbs & Evans, 1980; Grosz & Sidner, 1986; Pollack, 1986] whereas now there is progress on modeling collaborative plans with joint intentions [Grosz & Kraus, 1993; Lochbaum, 1994]. It is now accepted that the intentional approach is more appropriate for conversational dialog-based systems since the collaborative aspect of the dialog has to be captured and retained.

Present research using deep semantic techniques may employ a semantic interpreter which uses prepositions as its input propositions extracted by semantic concept detectors of a grammar-based sentence understanding unit. It then combines these propositions from multiple utterances to form larger units of meaning and must do this relative to the context in which the language was used.

In conversational dialog applications such as an intelligent tutoring system (ITS), where there is a need for a deep understanding of the semantics of language, hybrid techniques are used. These hybrid techniques combine statistical methods (e.g., Latent Semantic Analysis) for comparing student inputs with expected inputs to determine whether a question was answered correctly or not [e.g., Graesser et al., 1999] and the extraction of thematic roles based on the FrameNet [Baker, et al, 1998] from a student input [Gildea & Jurafsky, 2001].

The aforementioned cited articles include:

Appelt, D. & Pollack, M. (1990). Weighted abduction for plan ascription. Menlo Park, CA: SRI International. Technical Note 491.

Baker, Collin F., Fillmore, Charles J., and Lowe, John B. (1998): The Berkeley FrameNet project. *In Proceedings of the COLING-ACL*, Montreal, Canada.

Charniak, E. (1993). *Statistical Language Analysis*. Cambridge: Cambridge University Press.

Daniel Gildea and Daniel Jurafsky. 2002. Automatic Labeling of Semantic Roles. *Computational Linguistics* 28:3, 245-288.

DeJong, G. (1977). Skimming newspaper stories by computer. New Haven, CT: Department of Computer Science, Yale University. Research Report 104.

FrameNet: Theory and Practice. Christopher R. Johnson et al, *http://www.icsi.berkeley.edu/~framenet/book/book.html*

Graesser, A. C., Wiemer-Hastings, P., Wiemer-Hastings, K., Harter, D., Person, N., and the TRG (in press). Using latent semantic analysis to evaluate the contributions of students in AutoTutor. *Interactive Learning Environments.*

Graesser, A., Wiemer-Hastings, K., Wiemer-Hastings, P., Kreuz, R., & the Tutoring Research Group (2000). AutoTutor: A simulation of a human tutor, *Journal of Cognitive Systems Research*, 1,35-51.

Grice, H. P. (1969). Utterer's meaning and intentions. *Philosophical Review, 68*(2):147–177.

Grice, H. P. (1989). *Studies in the Ways of Words.* Cambridge, MA: Harvard University Press.

Grosz, B. & Kraus, S. (1993). Collaborative plans for group activities. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI '93), Chambery, France (vol. 1, pp. 367–373).

Grosz, B. J. & Sidner, C. L. (1986). Attentions, intentions and the structure of discourse. *Computational Linguistics, 12*, 175–204.

Hobbs, J. & Evans, D. (1980). Conversation as planned behavior. *Cognitive Science 4*(4), 349–377.

Hobbs, J. & Evans, D. (1980). Conversation as planned behavior. *Cognitive Science 4*(4), 349–377.

Hobbs, J., Stickel, M., Appelt, D., & Martin, P. (1993). Interpretation as abduction. *Artificial Intelligence 63*(1–2), 69–142.

Lascarides, A. & Asher, N. (1991). Discourse relations and defeasible knowledge. In Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL '91), Berkeley, CA (pp. 55–62).

Lascarides, A. & Oberlander, J. (1992). Temporal coherence and defeasible knowledge. *Theoretical Linguistics, 19*.

Lewis, D. (1979). Scorekeeping in a language game. *Journal of Philosophical Logic 6*, 339–359.

Litman, D.J., Pan, Shimei, Designing and evaluating an adaptive spoken dialogue system, User Modeling and User Adapted Interaction, 12, 2002.

Lochbaum, K. (1994). Using Collaborative Plans to Model the Intentional Structure of Discourse. PhD thesis, Harvard University.

McRoy, S. & Hirst, G. (1991). An abductive account of repair in conversation. AAAI Fall Symposium on Discourse Structure in Natural Language Understanding and Generation, Asilomar, CA (pp. 52–57).

Perrault, C. & Allen, J. (1980). A plan-based analysis of indirect speech acts. American Journal of *Computational Linguistics, 6*(3–4), 167–182.

Pollack, M. (1986). A model of plan inference that distinguishes between the beliefs of actors and observers. In Proceedings of 24th Annual Meeting of the Association for Computational Linguistics, New York (pp. 207–214).

Power, R. (1974). A Computer Model of Conversation. PhD. thesis, University of Edinburgh, Scotland.

Rayner, M. & Alshawi, H. (1992). Deriving database queries from logical forms by abductive definition expansion. In Proceedings of the Third Conference of Applied Natural Language Processing, Trento, Italy (pp. 1–8).

Reiger, C. (1974). Conceptual Memory: A Theory and Computer Program for Processing the Meaning Content of Natural Language Utterances. Stanford, CA: Stanford Artificial Intelligence Laboratory. Memo AIM-233.

Schank, R. (1975). *Conceptual Information Processing.* New York: Elsevier.

Schank, R., Lebowitz, M., & Birnbaum, L. (1980). An integrated understander. *American Journal of Computational Linguistics, 6*(1).

Searle, J. (1969). *Speech Acts: An Essay in the Philosophy of Language.* Cambridge: Cambridge University Press.

Sperber, D. & Wilson, D. (1986). *Relevance: Communication and Cognition.* Cambridge, MA: Harvard University Press.

The above are also incorporated by reference herein.

SUMMARY OF THE INVENTION

An object of the present invention, therefore, is to provide an improved system and method for overcoming the limitations of the prior art noted above;

A primary object of the present invention is to provide a word and phrase recognition system that is flexibly and optimally distributed across a client/platform computing architecture, so that improved accuracy, speed and uniformity can be achieved for a wide group of users;

A further object of the present invention is to provide a speech recognition system that efficiently integrates a distributed word recognition system with a natural language processing system, so that both individual words and entire speech utterances can be quickly and accurately recognized in any number of possible languages;

A related object of the present invention is to provide an efficient query response system so that an extremely accurate, real-time set of appropriate answers can be given in response to speech-based queries;

Yet another object of the present invention is to provide an interactive, real-time instructional/learning system that is distributed across a client/server architecture, and permits a real-time question/answer session with an interactive character;

A related object of the present invention is to implement such interactive character with an articulated response capability so that the user experiences a human-like interaction;

Still a further object of the present invention is to provide an INTERNET website with

speech processing capability so that voice based data and commands can be used to interact with such site, thus enabling voice-based e-commerce and e-support services to be easily scaleable;

Another object is to implement a distributed speech recognition system that utilizes environmental variables as part of the recognition process to improve accuracy and speed;

A further object is to provide a scaleable query/response database system, to support any number of query topics and users as needed for a particular application and instantaneous demand;

Yet another object of the present invention is to provide a query recognition system that employs a two-step approach, including a relatively rapid first step to narrow down the list of potential responses to a smaller candidate set, and a second more computationally intensive second step to identify the best choice to be returned in response to the query from the candidate set;

A further object of the present invention is to provide a natural language processing system that facilitates query recognition by extracting lexical components of speech utterances, which components can be used for rapidly identifying a candidate set of potential responses appropriate for such speech utterances;

Another related object of the present invention is to provide a natural language processing system that facilitates query recognition by comparing lexical components of speech utterances with a candidate set of potential response to provide an extremely accurate best response to such query.

Still another object of the present invention is to provide a natural language processing system which uses semantic decoding as part of a process for comprehending a question posed in a speech utterance;

One general aspect of the present invention, therefore, relates to a natural language query system (NLQS) that offers a fully interactive method for answering user's questions over a distributed network such as the INTERNET or a local intranet. This interactive system when implemented over the worldwide web (WWW) services of the INTERNET functions so that a client or user can ask a question in a natural language such as English, French, German or Spanish and receive the appropriate answer at his or her personal computer also in his or her native natural language.

The system is distributed and consists of a set of integrated software modules at the client's machine and another set of integrated software programs resident on a server or set of servers. The client-side software program is comprised of a speech recognition program, an agent and its control program, and a communication program. The server-side program is comprised of a communication program, a natural language engine (NLE), a database processor (DBProcess), an interface program

for interfacing the DBProcess with the NLE, and a SQL database. In addition, the client's machine is equipped with a microphone and a speaker. Processing of the speech utterance is divided between the client and server side so as to optimize processing and transmission latencies, and so as to provide support for even very thin client platforms.

In the context of an interactive learning application, the system is specifically used to provide a single-best answer to a user's question. The question that is asked at the client's machine is articulated by the speaker and captured by a microphone that is built in as in the case of a notebook computer or is supplied as a standard peripheral attachment. Once the question is captured, the question is processed partially by NLQS client-side software resident in the client's machine. The output of this partial processing is a set of speech vectors that are transported to the server via the INTERNET to complete the recognition of the user's questions. This recognized speech is then converted to text at the server.

After the user's question is decoded by the speech recognition engine (SRE) located at the server, the question is converted to a structured query language (SQL) query. This query is then simultaneously presented to a software process within the server called DBProcess for preliminary processing and to a Natural Language Engine (NLE) module for extracting the noun phrases (NP) of the user's question. During the process of extracting the noun phrase within the NLE, the tokens of the users' question are tagged. The tagged tokens are then grouped so that the NP list can be determined. This information is stored and sent to the DBProcess process.

In the DBProcess, the SQL query is fully customized using the NP extracted from the user's question and other environment variables that are relevant to the application. For example, in a training application, the user's selection of course, chapter and or section would constitute the environment variables. The SQL query is constructed using the extended SQL Full-Text predicates - **CONTAINS**, **FREETEXT**, **NEAR**, **AND**. The SQL query is next sent to the Full-Text search engine within the SQL database, where a Full-Text search procedure is initiated. The result of this search procedure is recordset of answers. This recordset contains stored questions that are similar linguistically to the user's question. Each of these stored questions has a paired answer stored in a separate text file, whose path is stored in a table of the database.

The entire recordset of returned stored answers is then returned to the NLE engine in the form of an array. Each stored question of the array is then linguistically processed sequentially one by one. This linguistic processing constitutes the second step of a 2-step algorithm to determine the single best answer to the user's question. This second step proceeds as follows: for each stored

question that is returned in the recordset, a NP of the stored question is compared with the NP of the user's question. After all stored questions of the array are compared with the user's question, the stored question that yields the maximum match with the user's question is selected as the best possible stored question that matches the user's question. The metric that is used to determine the best possible stored question is the number of noun phrases.

The stored answer that is paired to the best-stored question is selected as the one that answers the user's question. The ID tag of the question is then passed to the DBProcess. This DBProcess returns the answer which is stored in a file.

A communication link is again established to send the answer back to the client in compressed form. The answer once received by the client is decompressed and articulated to the user by the text-to-speech engine. Thus, the invention can be used in any number of different applications involving interactive learning systems, INTERNET related commerce sites, INTERNET search engines, etc.

Computer-assisted instruction environments often require the assistance of mentors or live teachers to answer questions from students. This assistance often takes the form of organizing a separate pre-arranged forum or meeting time that is set aside for chat sessions or live call-in sessions so that at a scheduled time answers to questions may be provided. Because of the time immediacy and the on-demand or asynchronous nature of on-line training where a student may log on and take instruction at any time and at any location, it is important that answers to questions be provided in a timely and cost-effective manner so that the user or student can derive the maximum benefit from the material presented.

This invention addresses the above issues. It provides the user or student with answers to questions that are normally channeled to a live teacher or mentor. This invention provides a single-best answer to questions asked by the student. The student asks the question in his or her own voice in the language of choice. The speech is recognized and the answer to the question is found using a number of technologies including distributed speech recognition, full-text search database processing, natural language processing and text-to-speech technologies. The answer is presented to the user, as in the case of a live teacher, in an articulated manner by an agent that mimics the mentor or teacher, and in the language of choice - English, French, German, Japanese or other natural spoken language. The user can choose the agent's gender as well as several speech parameters such as pitch, volume and speed of the character's voice.

Other applications that benefit from NLQS are e-commerce applications. In this application,

the user's query for a price of a book, compact disk or for the availability of any item that is to be purchased can be retrieved without the need to pick through various lists on successive web pages. Instead, the answer is provided directly to the user without any additional user input.

Similarly, it is envisioned that this system can be used to provide answers to frequently-asked questions (FAQs), and as a diagnostic service tool for e-support. These questions are typical of a give web site and are provided to help the user find information related to a payment procedure or the specifications of, or problems experienced with a product/service. In all of these applications, the NLQS architecture can be applied.

A number of inventive methods associated with these architectures are also beneficially used in a variety of INTERNET related applications.

Although the inventions are described below in a set of preferred embodiments, it will be apparent to those skilled in the art the present inventions could be beneficially used in many environments where it is necessary to implement fast, accurate speech recognition, and/or to provide a human-like dialog capability to an intelligent system.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a preferred embodiment of a natural language query system (NLQS) of the present invention, which is distributed across a client/server computing architecture, and can be used as an interactive learning system, an e-commerce system, an e-support system, and the like;

FIGs. 2A – 2C are a block diagram of a preferred embodiment of a client side system, including speech capturing modules, partial speech processing modules, encoding modules, transmission modules, agent control modules, and answer/voice feedback modules that can be used in the aforementioned NLQS;

FIG. 2D is a block diagram of a preferred embodiment of a set of initialization routines and procedures used for the client side system of FIG. 2A – 2C;

FIG. 3 is a block diagram of a preferred embodiment of a set of routines and procedures used for handling an iterated set of speech utterances on the client side system of FIG. 2A – 2C, transmitting speech data for such utterances to a remote server, and receiving appropriate responses back from such server;

FIG. 4 is a block diagram of a preferred embodiment of a set of initialization routines and procedures used for un-initializing the client side system of FIG. 2A – 2C;

FIG. 4A is a block diagram of a preferred embodiment of a set of routines and procedures used for implementing a distributed component of a speech recognition module for the server side system of FIG. 5;

FIG. 4B is a block diagram of a preferred set of routines and procedures used for implementing an SQL query builder for the server side system of FIG. 5;

FIG. 4C is a block diagram of a preferred embodiment of a set of routines and procedures used for implementing a database control process module for the server side system of FIG. 5;

FIG. 4D is a block diagram of a preferred embodiment of a set of routines and procedures used for implementing a natural language engine that provides query formulation support, a query response module, and an interface to the database control process module for the server side system of FIG. 5;

FIG. 5 is a block diagram of a preferred embodiment of a server side system, including a speech recognition module to complete processing of the speech utterances, environmental and grammar control modules, query formulation modules, a natural language engine, a database control module, and a query response module that can be used in the aforementioned NLQS;

FIG. 6 illustrates the organization of a full-text database used as part of server side system shown in FIG. 5;

FIG. 7A illustrates the organization of a full-text database course table used as part of server side system shown in FIG. 5 for an interactive learning embodiment of the present invention;

FIG. 7B illustrates the organization of a full-text database chapter table used as part of server side system shown in FIG. 5 for an interactive learning embodiment of the present invention;

FIG. 7C describes the fields used in a chapter table used as part of server side system shown in FIG. 5 for an interactive learning embodiment of the present invention;

FIG. 7D describes the fields used in a section table used as part of server side system shown in FIG. 5 for an interactive learning embodiment of the present invention;

FIG. 8 is a flow diagram of a first set of operations performed by a preferred embodiment of a natural language engine on a speech utterance including Tokenization, Tagging and Grouping;

FIG. 9 is a flow diagram of the operations performed by a preferred embodiment of a natural language engine on a speech utterance including stemming and Lexical Analysis

FIG. 10 is a block diagram of a preferred embodiment of a SQL database search and support system for the present invention;

FIGs. 11A–11C are flow diagrams illustrating steps performed in a preferred two step process implemented for query recognition by the NLQS of FIG. 2;

FIG. 12 is an illustration of another embodiment of the present invention implemented as part of a Web-based speech based learning/training System;

FIGs. 13–17 are illustrations of another embodiment of the present invention implemented as part of a Web-based e-commerce system;

FIG. 18 is an illustration of another embodiment of the present invention implemented as part of a voice-based Help Page for an E-Commerce Web Site.

FIG. 19 depicts a quasi-code implementation of an integrated speech processing method using both statistical and semantic processing in accordance with the present invention;

FIG. 20 illustrates a method for populating a speech lattice with semantic variants in accordance with the teachings of the present invention;

FIG. 21 illustrates a method for computing the closest semantic match between user articulated questions and stored semantic variants of the same.

DETAILED DESCRIPTION OF THE INVENTION

Overview

As alluded to above, the present inventions allow a user to ask a question in a natural language such as English, French, German, Spanish or Japanese at a client computing system (which can be as simple as a personal digital assistant or cell-phone, or as sophisticated as a high end desktop PC) and receive an appropriate answer from a remote server also in his or her native natural language. As such, the embodiment of the invention shown in FIG. 1 is beneficially used in what can be generally described as a Natural Language Query System (NLQS) 100, which is configured to interact on a real-time basis to give a human-like dialog capability/experience for e-commerce, e-support, and e-learning applications.

The processing for NLQS 100 is generally distributed across a client side system 150, a data link 160, and a server-side system 180. These components are well known in the art, and in a preferred embodiment include a personal computer system 150, an INTERNET connection 160A, 160B, and a larger scale computing system 180. It will be understood by those skilled in the art that these are merely exemplary components, and that the present invention is by no means limited to any particular implementation or combination of such systems. For example, client-side system 150

could also be implemented as a computer peripheral, a PDA, as part of a cell-phone, as part of an INTERNET-adapted appliance, an INTERNET linked kiosk, etc. Similarly, while an INTERNET connection is depicted for data link 160A, it is apparent that any channel that is suitable for carrying data between client system 150 and server system 180 will suffice, including a wireless link, an RF link, an IR link, a LAN, and the like. Finally, it will be further appreciated that server system 180 may be a single, large-scale system, or a collection of smaller systems interlinked to support a number of potential network users.

Initially speech input is provided in the form of a question or query articulated by the speaker at the client's machine or personal accessory as a speech utterance. This speech utterance is captured and partially processed by NLQS client-side software 155 resident in the client's machine. To facilitate and enhance the human-like aspects of the interaction, the question is presented in the presence of an animated character 157 visible to the user who assists the user as a personal information retriever/agent. The agent can also interact with the user using both visible text output on a monitor/display (not shown) and/or in audible form using a text to speech engine 159. The output of the partial processing done by SRE 155 is a set of speech vectors that are transmitted over communication channel 160 that links the user's machine or personal accessory to a server or servers via the INTERNET or a wireless gateway that is linked to the INTERNET as explained above. At server 180, the partially processed speech signal data is handled by a server-side SRE 182, which then outputs recognized speech text corresponding to the user's question. Based on this user question related text, a text-to-query converter 184 formulates a suitable query that is used as input to a database processor 186. Based on the query, database processor 186 then locates and retrieves an appropriate answer using a customized SQL query from database 188. A Natural Language Engine 190 facilitates structuring the query to database 188. After a matching answer to the user's question is found, the former is transmitted in text form across data link 160B, where it is converted into speech by text to speech engine 159, and thus expressed as oral feedback by animated character agent 157.

Because the speech processing is broken up in this fashion, it is possible to achieve real-time, interactive, human-like dialog consisting of a large, controllable set of questions/answers. The assistance of the animated agent 157 further enhances the experience, making it more natural and comfortable for even novice users. To make the speech recognition process more reliable, context-specific grammars and dictionaries are used, as well as natural language processing routines at NLE 190, to analyze user questions lexically. While context-specific processing of speech data is known

in the art (*see e.g.*, U.S. Patent Nos. 5,960,394, 5,867,817, 5,758,322 and 5,384,892 incorporated by reference herein) the present inventors are unaware of any such implementation as embodied in the present inventions. The text of the user's question is compared against text of other questions to identify the question posed by the user by DB processor/engine (DBE) 186. By optimizing the interaction and relationship of the SR engines 155 and 182, the NLP routines 190, and the dictionaries and grammars, an extremely fast and accurate match can be made, so that a unique and responsive answer can be provided to the user.

On the server side 180, interleaved processing further accelerates the speech recognition process. In simplified terms, the query is presented simultaneously both to NLE 190 after the query is formulated, as well as to DBE 186. NLE 190 and SRE 182 perform complementary functions in the overall recognition process. In general, SRE 182 is primarily responsible for determining the identity of the words articulated by the user, while NLE 190 is responsible for the linguistic morphological analysis of both the user's query and the search results returned after the database query.

After the user's query is analyzed by NLE 190 some parameters are extracted and sent to the DBProcess. Additional statistics are stored in an array for the 2nd step of processing. During the 2nd step of 2-step algorithm, the recordset of preliminary search results are sent to the NLE 160 for processing. At the end of this 2nd step, the single question that matches the user's query is sent to the DBProcess where further processing yields the paired answer that is paired with the single best stored question.

Thus, the present invention uses a form of natural language processing (NLP) to achieve optimal performance in a speech based web application system. While NLP is known in the art, prior efforts in Natural Language Processing (NLP) work nonetheless have not been well integrated with Speech Recognition (SR) technologies to achieve reasonable results in a web-based application environment. In speech recognition, the result is typically a lattice of possible recognized words each with some probability of fit with the speech recognizer. As described before, the input to a typical NLP system is typically a large linguistic unit. The NLP system is then charged with the parsing, understanding and indexing of this large linguistic unit or set of transcribed utterances. The result of this NLP process is to understand lexically or morphologically the entire linguistic unit as opposed to word recognition. Put another way, the linguistic unit or sentence of connected words output by the SRE has to be understood lexically, as opposed to just being "recognized".

As indicated earlier, although speech recognition technology has been available for several years, the technical requirements for the NLQS invention have placed severe restrictions on the specifications for the speech recognition accuracy that is required for an application that combines speech recognition and natural language processing to work satisfactorily. In realizing that even with the best of conditions, it might be not be possible to achieve the perfect 100% speech recognition accuracy that is required, the present invention employs an algorithm that balances the potential risk of the speech recognition process with the requirements of the natural language processing so that even in cases where perfect speech recognition accuracy is not achieved for each word in the query, the entire query itself is nonetheless recognized with sufficient accuracy.

This recognition accuracy is achieved even while meeting very stringent user constraints, such as short latency periods of 3 to 5 seconds (ideally – ignoring transmission latencies which can vary) for responding to a speech-based query, and for a potential set of 100 – 250 query questions. This quick response time gives the overall appearance and experience of a real-time discourse that is more natural and pleasant from the user's perspective. Of course, non-real time applications, such as translation services for example, can also benefit from the present teachings as well, since a centralized set of HMMs, grammars, dictionaries, etc., are maintained.

General Aspects of Speech Recognition Used in the Present Inventions

General background information on speech recognition can be found in the prior art references discussed above and incorporated by reference herein. Nonetheless, a discussion of some particular exemplary forms of speech recognition structures and techniques that are well-suited for NLQS 100 is provided next to better illustrate some of the characteristics, qualities and features of the present inventions.

Speech recognition technology is typically of two types – speaker independent and speaker dependent. In speaker-dependent speech recognition technology, each user has a voice file in which a sample of each potentially recognized word is stored. Speaker-dependent speech recognition systems typically have large vocabularies and dictionaries making them suitable for applications as dictation and text transcribing. It follows also that the memory and processor resource requirements for the speaker-dependent can be and are typically large and intensive.

Conversely speaker-independent speech recognition technology allows a large group of users to use a single vocabulary file. It follows then that the degree of accuracy that can be achieved is a function of the size and complexity of the grammars and dictionaries that can be supported for a

given language. Given the context of applications for which NLQS, the use of small grammars and dictionaries allow speaker independent speech recognition technology to be implemented in NLQS.

The key issues or requirements for either type – speaker-independent or speaker-dependent, are accuracy and speed. As the size of the user dictionaries increase, the speech recognition accuracy metric – word error rate (WER) and the speed of recognition decreases. This is so because the search time increases and the pronunciation match becomes more complex as the size of the dictionary increases.

The basis of the NLQS speech recognition system is a series of Hidden Markov Models (HMM), which, as alluded to earlier, are mathematical models used to characterize any time varying signal. Because parts of speech are considered to be based on an underlying sequence of one or more symbols, the HMM models corresponding to each symbol are trained on vectors from the speech waveforms. The Hidden Markov Model is a finite set of states, each of which is associated with a (generally multi-dimensional) probability distribution. Transitions among the states are governed by a set of probabilities called transition probabilities. In a particular state an outcome or observation can be generated, according to an associated probability distribution. This finite state machine changes state once every time unit, and each time **t** such that a state **j** is entered, a spectral parameter vector $\mathbf{O}_t$ is generated with probability density $\mathbf{B_j(O_t)}$. It is only the outcome, not the state which is visible to an external observer and therefore states are ``hidden" to the outside; hence the name Hidden Markov Model.

In isolated speech recognition, it is assumed that the sequence of observed speech vectors corresponding to each word can each be described by a Markov model as follows:

$$\mathbf{O} = \mathbf{o}_1, \mathbf{o}_2, \ldots\ldots\ldots \mathbf{o}_T \qquad (1\text{-}1)$$

where $\mathbf{o}_t$ is a speech vector observed at time t. The isolated word recognition then is to compute:

$$\arg\max \{ P(w_i | \mathbf{O})\} \qquad (1\text{-}2)$$

By using Bayes' Rule,

$$\{ P(w_i | \mathbf{O})\} = [P(\mathbf{O} | w_i) P(w_i)] / P(\mathbf{O}) \qquad (1\text{-}3)$$

In the general case, the Markov model when applied to speech also assumes a finite state machine which changes state once every time unit and each time that a state j is entered, a speech vector $\mathbf{o}_t$ is generated from the probability density $b_j(\mathbf{o}_t)$. Furthermore, the transition from state i to state j is also probabilistic and is governed by the discrete probability a_{ij}.

For a state sequence X, the joint probability that **O** is generated by the model M moving

through a state sequence X is the product of the transition probabilities and the output probabilities. Only the observation sequence is known – the state sequence is hidden as mentioned before.

Given that X is unknown, the required likelihood is computed by summing over all possible state sequences X = x(1), x(2), x(3) ,….. x(T), that is

$$P(\mathbf{O}|M) = \Sigma \{ a_{x(0)\,x(1)} \Pi\, b(x)\,(\mathbf{o}_t)\, a_{x(t)\,x(t+1)} \}$$

Given a set of models M_i, corresponding to words w_i equation 1-2 is solved by using 1-3 and also by assuming that:

$$P(\mathbf{O}|w_i) = P(\mathbf{O}|M_i)$$

All of this assumes that the parameters $\{a_{ij}\}$ and $\{b_j(o_t)\}$ are known for each model M_i. This can be done, as explained earlier, by using a set of training examples corresponding to a particular model. Thereafter, the parameters of that model can be determined automatically by a robust and efficient re-estimation procedure. So if a sufficient number of representative examples of each word are collected, then a HMM can be constructed which simply models all of the many sources of variability inherent in real speech. This training is well-known in the art, so it is not described at length herein, except to note that the distributed architecture of the present invention enhances the quality of HMMs, since they are derived and constituted at the server side, rather than the client side. In this way, appropriate samples from users of different geographical areas can be easily compiled and analyzed to optimize the possible variations expected to be seen across a particular language to be recognized. Uniformity of the speech recognition process is also well-maintained, and error diagnostics are simplified, since each prospective user is using the same set of HMMs during the recognition process.

To determine the parameters of a HMM from a set of training samples, the first step typically is to make a rough guess as to what they might be. Then a refinement is done using the Baum-Welch estimation formulae. By these formulae, the maximum likelihood estimates of μ_j (where μ_j is mean vector and Σ_j is covariance matrix) is:

$$\mu_j = \Sigma^{T}_{t=1} L_j(t)\, o_t \,/\, [\Sigma^{T}_{t=1} L_j(t)\, o_t]$$

A forward-backward algorithm is next used to calculate the probability of state occupation $L_j(t)$. If the forward probability $\alpha_j(t)$ for some model M with N states is defined as:

$$\alpha_j(t) = P(o_1, \ldots\ldots, o_t, x(t) = j|M)$$

This probability can be calculated using the recursion:

$$\alpha_j(t) = [\Sigma^{N-1}_{i=2}\, \alpha(t-1)\, a_{ij}] b_j(o_t)$$

Similarly the backward probability can be computed using the recursion:

$$\beta_i(t) = \Sigma^{N-1}_{j=2}\, a_{ij}\, b_j(o_{t+1})(t+1)$$

Realizing that the forward probability is a joint probability and the backward probability is a conditional probability, the probability of state occupation is the product of the two probabilities:

$$\alpha_j(t)\,\beta_j(t) = P(\mathbf{O}, x(t) = j \mid M)$$

Hence the probability of being in state j at a time t is:

$$L_j(t) = 1/P\,[\alpha_j(t)\,\beta_j(t)]$$

where $P = P(\mathbf{O} \mid M)$

To generalize the above for continuous speech recognition, we assume the maximum likelihood state sequence where the summation is replaced by a maximum operation. Thus for a given model M, let $\phi_j(t)$ represent the maximum likelihood of observing speech vectors o_1 **to** o_t and being used in state j at time t:

$$\phi_j(t) = \max\{\phi_j(t)(t-1)\alpha_{ij}\}\,\beta_j(o_t)$$

Expressing this logarithmically to avoid underflow, this likelihood becomes:

$$\psi_j(t) = \max\{\psi_i(t-1) + \log(\alpha_{ij})\} + \log(b_j(o_t))$$

This is also known as the Viterbi algorithm. It can be visualized as finding the best path through a matrix where the vertical dimension represents the states of the HMM and horizontal dimension represents frames of speech i.e. time. To complete the extension to connected speech recognition, it is further assumed that each HMM representing the underlying sequence is connected. Thus the training data for continuous speech recognition should consist of connected utterances; however, the boundaries between words do not have to be known.

To improve computational speed/efficiency, the Viterbi algorithm is sometimes extended to achieve convergence by using what is known as a Token Passing Model. The token passing model represents a partial match between the observation sequence o_1 **to** o_t and a particular model, subject to the constraint that the model is in state j at time t. This token passing model can be extended easily to connected speech environments as well if we allow the sequence of HMMs to be defined as a finite state network. A composite network that includes both phoneme-based HMMs and complete words can be constructed so that a single-best word can be recognized to form connected speech using word N-best extraction from the lattice of possibilities. This composite form of HMM-based connected speech recognizer is the basis of the NLQS speech recognizer module. Nonetheless, the present invention is not limited as such to such specific forms of speech

recognizers, and can employ other techniques for speech recognition if they are otherwise compatible with the present architecture and meet necessary performance criteria for accuracy and speed to provide a real-time dialog experience for users.

The representation of speech for the present invention's HMM-based speech recognition system assumes that speech is essentially either a quasi-periodic pulse train (for voiced speech sounds) or a random noise source (for unvoiced sounds). It may be modeled as two sources – one a impulse train generator with pitch period P and a random noise generator which is controlled by a voice/unvoiced switch. The output of the switch is then fed into a gain function estimated from the speech signal and scaled to feed a digital filter H(z) controlled by the vocal tract parameter characteristics of the speech being produced. All of the parameters for this model – the voiced/unvoiced switching, the pitch period for voiced sounds, the gain parameter for the speech signal and the coefficient of the digital filter, vary slowly with time. In extracting the acoustic parameters from the user's speech input so that it can evaluated in light of a set of HMMs, cepstral analysis is typically used to separate the vocal tract information from the excitation information. The cepstrum of a signal is computed by taking the Fourier (or similar) transform of the log spectrum. The principal advantage of extracting cepstral coefficients is that they are de-correlated and the diagonal covariances can be used with HMMs. Since the human ear resolves frequencies non-linearly across the audio spectrum, it has been shown that a front-end that operates in a similar non-linear way improves speech recognition performance.

Accordingly, instead of a typical linear prediction-based analysis, the front-end of the NLQS speech recognition engine implements a simple, fast Fourier transform based filter bank designed to give approximately equal resolution on the Mel-scale. To implement this filter bank, a window of speech data (for a particular time frame) is transformed using a software based Fourier transform and the magnitude taken. Each FFT magnitude is then multiplied by the corresponding filter gain and the results accumulated. The cepstral coefficients that are derived from this filter-bank analysis at the front end are calculated during a first partial processing phase of the speech signal by using a Discrete Cosine Transform of the log filter bank amplitudes. These cepstral coefficients are called Mel-Frequency Cepstral Coefficients (MFCC) and they represent some of the speech parameters transferred from the client side to characterize the acoustic features of the user's speech signal. These parameters are chosen for a number of reasons, including the fact that they can be quickly and consistently derived even across systems of disparate capabilities (i.e., for everything from a low power PDA to a high powered desktop system), they give good discrimination, they lend themselves

to a number of useful recognition related manipulations, and they are relatively small and compact in size so that they can be transported rapidly across even a relatively narrow band link. Thus, these parameters represent the least amount of information that can be used by a subsequent server side system to adequately and quickly complete the recognition process.

To augment the speech parameters an energy term in the form of the logarithm of the signal energy is added. Accordingly, RMS energy is added to the 12 MFCC's to make 13 coefficients. These coefficients together make up the partially processed speech data transmitted in compressed form from the user's client system to the remote server side.

The performance of the present speech recognition system is enhanced significantly by computing and adding time derivatives to the basic static MFCC parameters at the server side. These two other sets of coefficients -- the delta and acceleration coefficients representing change in each of the 13 values from frame to frame (actually measured across several frames), are computed during a second partial speech signal processing phase to complete the initial processing of the speech signal, and are added to the original set of coefficients after the latter are received. These MFCCs together with the delta and acceleration coefficients constitute the observation vector $\mathbf{O}_t$ mentioned above that is used for determining the appropriate HMM for the speech data.

The delta and acceleration coefficients are computed using the following regression formula:

$$d_t = \Sigma^{\theta}_{\theta=1} [c_{t+\theta} - c_{t-\theta}] / 2\Sigma^{\theta}_{\theta=1}\theta^2$$

where d_t is a delta coefficient at time t computed in terms of the corresponding static coefficients:

$$d_t = [c_{t+\theta} - c_{t-\theta}] / 2\theta$$

In a typical stand-alone implementation of a speech recognition system, the entire SR engine runs on a single client. In other words, both the first and second partial processing phases above are executed by the same DSP (or microprocessor) running a ROM or software code routine at the client's computing machine.

In contrast, because of several considerations, specifically - cost, technical performance, and client hardware uniformity, the present NLQS system uses a partitioned or distributed approach. While some processing occurs on the client side, the main speech recognition engine runs on a centrally located server or number of servers. More specifically, as noted earlier, capture of the speech signals, MFCC vector extraction and compression are implemented on the client's machine during a first partial processing phase. The routine is thus streamlined and simple enough to be

implemented within a browser program (as a plug in module, or a downloadable applet for example) for maximum ease of use and utility. Accordingly, even very "thin" client platforms can be supported, which enables the use of the present system across a greater number of potential sites. The primary MFCCs are then transmitted to the server over the channel, which, for example, can include a dial-up INTERNET connection, a LAN connection, a wireless connection and the like. After decompression, the delta and acceleration coefficients are computed at the server to complete the initial speech processing phase, and the resulting observation vectors $\mathbf{O}_t$ are also determined.

General Aspects of Speech Recognition Engine

The speech recognition engine is also located on the server, and is based on a HTK-based recognition network compiled from a word-level network, a dictionary and a set of HMMs. The recognition network consists of a set of nodes connected by arcs. Each node is either a HMM model instance or a word end. Each model node is itself a network consisting of states connected by arcs. Thus when fully compiled, a speech recognition network consists of HMM states connected by transitions. For an unknown input utterance with T frames, every path from the start node to the exit node of the network passes through T HMM states. Each of these paths has log probability which is computed by summing the log probability of each individual transition in the path and the log probability of each emitting state generating the corresponding observation. The function of the Viterbi decoder is find those paths through the network which have the highest log probability. This is found using the Token Passing algorithm. In a network that has many nodes, the computation time is reduced by only allowing propagation of those tokens which will have some chance of becoming winners. This process is called pruning.

Natural Language Processor

In a typical natural language interface to a database, the user enters a question in his/her natural language, for example, English. The system parses it and translates it to a query language expression. The system then uses the query language expression to process the query and if the search is successful, a recordset representing the results is displayed in English either formatted as raw text or in a graphical form. For a natural language interface to work well involves a number of technical requirements.

For example, it needs to be robust – in the sentence '*What's the departments turnover*' it needs to decide that the word *whats* = *what's* = *what is*. And it also has to determine that *departments* =

department's. In addition to being robust, the natural language interface has to distinguish between the several possible forms of ambiguity that may exist in the natural language – lexical, structural, reference and ellipsis ambiguity. All of these requirements, in addition to the general ability to perform basic linguistic morphological operations of tokenization, tagging and grouping, are implemented within the present invention.

Tokenization is implemented by a text analyzer which treats the text as a series of tokens or useful meaningful units that are larger than individual characters, but smaller than phrases and sentences. These include words, separable parts of words, and punctuation. Each token is associated with an offset and a length. The first phase of tokenization is the process of segmentation which extracts the individual tokens from the input text and keeps track of the offset where each token originated in the input text. The tokenizer output lists the offset and category for each token. In the next phase of the text analysis, the tagger uses a built-in morphological analyzer to look up each word/token in a phrase or sentence and internally lists all parts of speech. The output is the input string with each token tagged with a parts of speech notation. Finally the grouper which functions as a phrase extractor or phrase analyzer, determines which groups of words form phrases. These three operations which are the foundations for any modern linguistic processing schemes, are fully implemented in optimized algorithms for determining the single-best possible answer to the user's question.

SQL Database and Full-Text Query

Another key component of present system is a SQL-database. This database is used to store text, specifically the answer-question pairs are stored in full-text tables of the database. Additionally, the full-text search capability of the database allows full-text searches to be carried out.

While a large portion of all digitally stored information is in the form of unstructured data, primarily text, it is now possible to store this textual data in traditional database systems in character-based columns such as **varchar** and **text**. In order to effectively retrieve textual data from the database, techniques have to be implemented to issue queries against textual data and to retrieve the answers in a meaningful way where it provides the answers as in the case of the NLQS system.

There are two major types of textual searches: Property - This search technology first applies filters to documents in order to extract properties such as author, subject, type, word count, printed page count, and time last written, and then issues searches against those properties; Full-text –this

search technology first creates indexes of all non-noise words in the documents, and then uses these indexes to support linguistic searches and proximity searches.

Two additional technologies are also implemented in this particular RDBMs: SQL Server also have been integrated: A Search service - a full-text indexing and search service that is called both index engine and search, and a parser that accepts full-text SQL extensions and maps them into a form that can be processed by the search engine.

The four major aspects involved in implementing full-text retrieval of plain-text data from a full-text-capable database are: Managing the definition of the tables and columns that are registered for full-text searches; Indexing the data in registered columns - the indexing process scans the character streams, determines the word boundaries (this is called word breaking), removes all noise words (this also is called stop words), and then populates a full-text index with the remaining words; Issuing queries against registered columns for populated full-text indexes; Ensuring that subsequent changes to the data in registered columns gets propagated to the index engine to keep the full-text indexes synchronized.

The underlying design principle for the indexing, querying, and synchronizing processes is the presence of a full-text unique key column (or single-column primary key) on all tables registered for full-text searches. The full-text index contains an entry for the non-noise words in each row together with the value of the key column for each row.

When processing a full-text search, the search engine returns to the database the key values of the rows that match the search criteria.

The full-text administration process starts by designating a table and its columns of interest for full-text search. Customized NLQS stored procedures are used first to register tables and columns as eligible for full-text search. After that, a separate request by means of a stored procedure is issued to populate the full-text indexes. The result is that the underlying index engine gets invoked and asynchronous index population begins. Full-text indexing tracks which significant words are used and where they are located. For example, a full-text index might indicate that the word "NLQS" is found at word number 423 and word number 982 in the **Abstract** column of the **DevTools** table for the row associated with a **ProductID** of 6. This index structure supports an efficient search for all items containing indexed words as well as advanced search operations, such as phrase searches and proximity searches. (An example of a phrase search is looking for "*white elephant*," where "*white*" is followed by "*elephant*". An example of a proximity search is looking for "*big*" and "*house*" where "*big*" occurs near "*house*".) To prevent the full-text index from becoming

bloated, noise words such as "*a*," "*and*," and "*the*" are ignored.

Extensions to the Transact-SQL language are used to construct full-text queries. The two key predicates that are used in the NLQS are **CONTAINS** and **FREETEXT.**

The **CONTAINS** predicate is used to determine whether or not values in full-text registered columns contain certain words and phrases. Specifically, this predicate is used to search for:

- A word or phrase.
- The prefix of a word or phrase.
- A word or phrase that is near another.
- A word that is an inflectional form of another (for example, "*drive*" is the inflectional stem of "*drives*," "*drove*," "*driving*," and "*driven*").
- A set of words or phrases, each of which is assigned a different weighting.

The relational engine within SQL Server recognizes the **CONTAINS** and **FREETEXT** predicates and performs some minimal syntax and semantic checking, such as ensuring that the column referenced in the predicate has been registered for full-text searches. During query execution, a full-text predicate and other relevant information are passed to the full-text search component. After further syntax and semantic validation, the search engine is invoked and returns the set of unique key values identifying those rows in the table that satisfy the full-text search condition. In addition to the **FREETEXT** and **CONTAINS**, other predicates such as **AND**, **LIKE**, **NEAR** are combined to create the customized NLQS SQL construct.

Full-Text Query Architecture of the SQL Database

The full-text query architecture is comprised of the following several components – Full-Text Query component, the SQL Server Relational Engine, the Full-Text provider and the Search Engine.

The **Full-Text Query** component of the SQL database accept a full-text predicate or rowset-valued function from the SQL Server; transform parts of the predicate into an internal format, and sends it to Search Service, which returns the matches in a rowset. The rowset is then sent back to SQL Server. SQL Server uses this information to create the resultset that is then returned to the submitter of the query.

The **SQL Server Relational Engine** accepts the **CONTAINS** and **FREETEXT** predicates as well as the **CONTAINSTABLE()** and **FREETEXTTABLE()** rowset-valued functions. During parse time, this code checks for conditions such as attempting to query a column

that has not been registered for full-text search. If valid, then at run time, the ft_search_condition and context information is sent to the full-text provider. Eventually, the full-text provider returns a rowset to SQL Server, which is used in any joins (specified or implied) in the original query. The **Full-Text Provider** parses and validates the ft_search_condition, constructs the appropriate internal representation of the full-text search condition, and then passes it to the search engine. The result is returned to the relational engine by means of a rowset of rows that satisfy ft_search_condition.

Client Side System 150

Client Side System 150

The architecture of client-side system 150 of Natural Language Query System 100 is illustrated in greater detail in FIGs. 2A – 2C. Referring to FIG. 2A, the three main processes effectuated by Client System 150 are illustrated as follows: Initialization process 200A consisting of SRE 201, Communication 202 and Microsoft (MS) Agent 203 routines; at FIG. 2B an iterative process 200B consisting of two sub-routines: a) Receive User Speech 208 – made up of SRE 204 and Communication 205; and b) Receive Answer from Server 207 – made up of MS Speak Agent 206, Communication 209, Voice data file 210 and Text to Speech Engine 211. Finally, in FIG. 2C un-initialization process 200C is made up of three sub-routines: SRE 212, Communication 213, and MS Agent 214. Each of the above three processes are described in detail in the following paragraphs. It will be appreciated by those skilled in the art that the particular implementation for such processes and routines will vary from client platform to platform, so that in some environments such processes may be embodied in hard-coded routines executed by a dedicated DSP, while in others they may be embodied as software software routines executed by a shared host processor, and in still others a combination of the two may be used.

Initialization at Client System 150

The initialization of the Client System 150 is illustrated in FIG. 2D and is comprised generally of 3 separate initializing processes: client-side Speech Recognition Engine 220A, MS Agent 220B and Communication processes 220C.

Initialization of Speech Recognition Engine 220A

Speech Recognition Engine 155 is initialized and configured using the routines shown in 220A. First, an SRE COM Library is initialized. Next, memory **220** is allocated to hold Source and

Coder objects, are created by a routine 221. . Loading of configuration file **221A** from configuration data file **221B** also takes place at the same time that the SRE Library is initialized. In configuration file **221B**, the type of the input of Coder and the type of the output of the Coder are declared. The structure, operation, etc. of such routines are well-known in the art, and they can be implemented using a number of fairly straightforward approaches. Accordingly, they are not discussed in detail herein. Next, Speech and Silence components of an utterance are calibrated using a routine **222**, in a procedure that is also well-known in the art. To calibrate the speech and silence components, the user preferably articulates a sentence that is displayed in a text box on the screen. The SRE library then estimates the noise and other parameters required to find e silence and speech elements of future user utterances.

Initialization of MS Agent 220B

The software code used to initialize and set up a MS Agent 220B is also illustrated in FIG. 2D. The MS Agent 220B routine is responsible for coordinating and handling the actions of the animated agent 157 (FIG.1). This initialization thus consists of the following steps:

1. Initialize COM library **223**. This part of the code initializes the COM library, which is required to use ActiveX Controls, which controls are well-known in the art.
2. Create instance of Agent Server **224** - this part of the code creates an instance of Agent ActiveX control.
3. Loading of MS Agent **225** - this part of the code loads MS Agent character from a specified file **225A** containing general parameter data for the Agent Character, such as the overall appearance, shape, size, etc.
4. Get Character Interface **226** - this part of the code gets an appropriate interface for the specified character; for example, characters may have different control/interaction capabilities that can be presented to the user.
5. Add Commands to Agent Character Option **227** - this part of the code adds commands to an Agent Properties sheet, which sheet can be accessed by clicking on the icon that appears in the system tray, when the Agent character is loaded *e.g.*, that the character can Speak, how he/she moves, TTS Properties, etc.
6. Show the Agent Character **228** - this part of the code displays the Agent character on the screen so it can be seen by the user;
7. AgentNotifySink - to handle events. This part of the code creates AgentNotifySink object

229, registers it at **230** and then gets the Agent Properties interface **231**. The property sheet for the Agent character is assigned using routine **232**.

8. Do Character Animations **233** - This part of the code plays specified character animations to welcome the user to NLQS 100.

The above then constitutes the entire sequence required to initialize the MS Agent. As with the SRE routines, the MS Agent routines can be implemented in any suitable and conventional fashion by those skilled in the art based on the present teachings. The particular structure, operation, etc. of such routines is not critical, and thus they are not discussed in detail herein.

In a preferred embodiment, the MS Agent is configured to have an appearance and capabilities that are appropriate for the particular application. For instance, in a remote learning application, the agent has the visual form and mannerisms/attitude/gestures of a college professor. Other visual props (blackboard, textbook, etc.) may be used by the agent and presented to the user to bring to mind the experience of being in an actual educational environment. The characteristics of the agent may be configured at the client side 150, and/or as part of code executed by a browser program (not shown) in response to configuration data and commands from a particular web page. For example, a particular website offering medical services may prefer to use a visual image of a doctor. These and many other variations will be apparent to those skilled in the art for enhancing the human-like, real-time dialog experience for users.

Initialization of Communication Link 160A

Initialization of Communication Link 160A

The initialization of Communication Link 160A is shown with reference to process 220C FIG. 2D. Referring to FIG. 2D, this initialization consists of the following code components: Open INTERNET Connection **234** - this part of the code opens an INTERNET Connection and sets the parameter for the connection. Then Set Callback Status routine **235** sets the callback status so as to inform the user of the status of connection. Finally Start New HTTP INTERNET Session **236** starts a new INTERNET session. The details of Communications Link 160 and the set up process 220C are not critical, and will vary from platform to platform. Again, in some cases, users may use a low-speed dial-up connection, a dedicated high speed switched connection (T1 for example), an always-on xDSL connection, a wireless connection, and the like.

Iterative Processing of Queries/Answers

As illustrated in **FIG. 3**, once initialization is complete, an iterative query/answer process is launched when the user presses the Start Button to initiate a query. Referring to **FIG. 3**, the iterative query/answer process consists of two main sub-processes implemented as routines on the client side system 150: **Receive User Speech 240** and **Receive User Answer 243.** The **Receive User Speech 240** routine receives speech from the user (or another audio input source), while the **Receive User Answer 243** routine receives an answer to the user's question in the form of text from the server so that it can be converted to speech for the user by text-to-speech engine 159. As used herein, the term "query" is referred to in the broadest sense to refer, to either a question, a command, or some form of input used as a control variable by the system. For example, a query may consist of a question directed to a particular topic, such as "what is a network" in the context of a remote learning application. In an e-commerce application a query might consist of a command to "list all books by Mark Twain" for example. Similarly, while the answer in a remote learning application consists of text that is rendered into audible form by the text to speech engine 159, it could also be returned as another form of multi-media information, such as a graphic image, a sound file, a video file, etc. depending on the requirements of the particular application. Again, given the present teachings concerning the necessary structure, operation, functions, performance, etc., of the client-side Receive User Speech 240 and Receiver User Answer 243 routines, one of ordinary skill in the art could implement such in a variety of ways.

Receive User Speech – As illustrated in **FIG. 3**, the Receive User Speech routine 240 consists of a SRE **241** and a Communication **242** process, both implemented again as routines on the client side system 150 for receiving and partially processing the user's utterance. SRE routine 241 uses a coder **248** which is prepared so that a coder object receives speech data from a source object. Next the Start Source **249** routine is initiated. This part of the code initiates data retrieval using the source Object which will in turn be given to the Coder object. Next, MFCC vectors **250** are extracted from the Speech utterance continuously until silence is detected. As alluded to earlier, this represents the first phase of processing of the input speech signal, and in a preferred embodiment, it is intentionally restricted to merely computing the MFCC vectors for the reasons already expressed above. These vectors include the 12 cepstral coefficients and the RMS energy term, for a total of 13 separate numerical values for the partially processed speech signal.

In some environments, nonetheless, it is conceivable that the MFCC delta parameters and MFCC acceleration parameters can also be computed at client side system 150, depending on the computation resources available, the transmission bandwidth in data link 160A available to server side system 180, the speed of a transceiver used for carrying data in the data link, etc. These parameters can be determined automatically by client side system upon initializing SRE 155 (using some type of calibration routine to measure resources), or by direct user control, so that the partitioning of signal processing responsibilities can be optimized on a case-by-case basis. In some applications, too, server side system 180 may lack the appropriate resources or routines for completing the processing of the speech input signal. Therefore, for some applications, the allocation of signal processing responsibilities may be partitioned differently, to the point where in fact both phases of the speech signal processing may take place at client side system 150 so that the speech signal is completely - rather than partially –processed and transmitted for conversion into a query at server side system 180.

Again in a preferred embodiment, to ensure reasonable accuracy and real-time performance from a query/response perspective, sufficient resources are made available in a client side system so that 100 frames per second of speech data can be partially processed and transmitted through link 160A. Since the least amount of information that is necessary to complete the speech recognition process (only 13 coefficients) is sent, the system achieves a real-time performance that is believed to be highly optimized, because other latencies (i.e., client-side computational latencies, packet formation latencies, transmission latencies) are minimized. It will be apparent that the principles of the present invention can be extended to other SR applications where some other methodology is used for breaking down the speech input signal by an SRE (i.e., non-MFCC based). The only criteria is that the SR processing be similarly dividable into multiple phases, and with the responsibility for different phases being handled on opposite sides of link 160A depending on overall system performance goals, requirements and the like. This functionality of the present invention can thus be achieved on a system-by-system basis, with an expected and typical amount of optimization being necessary for each particular implementation.

Thus, the present invention achieves a response rate performance that is tailored in accordance with the amount of information that is computed, coded and transmitted by the client side system 150. So in applications where real-time performance is most critical, the least possible amount of extracted speech data is transmitted to reduce these latencies, and, in other applications, the amount of extracted speech data that is processed, coded and transmitted can be varied.

Communication – transmit communication module **242** is used to implement the transport of data from the client to the server over the data link 160A, which in a preferred embodiment is the INTERNET. As explained above, the data consists of encoded MFCC vectors that will be used at then server-side of the Speech Recognition engine to complete the speech recognition decoding. The sequence of the communication is as follows:

OpenHTTPRequest 251 - this part of the code first converts MFCC vectors to a stream of bytes, and then processes the bytes so that it is compatible with a protocol known as HTTP. This protocol is well-known in the art, and it is apparent that for other data links another suitable protocol would be used.

1. **Encode MFCC Byte Stream 251** - this part of the code encodes the MFCC vectors, so that they can be sent to the server via HTTP.
2. **Send data 252** - this part of the code sends MFCC vectors to the server using the INTERNET connection and the HTTP protocol.

Wait for the Server Response 253 - this part of the code monitors the data link 160A a response from server side system 180 arrives. In summary, the MFCC parameters are extracted or observed on-the-fly from the input speech signal. They are then encoded to a HTTP byte stream and sent in a streaming fashion to the server before the silence is detected – i.e. sent to server side system 180 before the utterance is complete. This aspect of the invention also facilitates a real-time behavior, since data can be transmitted and processed even while the user is still speaking.

Receive Answer from Server 243 is comprised of the following modules as shown in **FIG. 3**.: MS Agent **244**, Text-to-Speech Engine **245** and receive communication modules **246**. All three modules interact to receive the answer from server side system 180. As illustrated in **FIG. 3**, the receive communication process consists of three separate processes implemented as a receive routine on client side system 150: a Receive the Best Answer **258** receives the best answer over data link 160B (the HTTP communication channel). The answer is de-compressed at 259 and then the

answer is passed by code 260 to the MS Agent 244, where it is received by code portion 254. A routine 255 then articulates the answer using text-to-speech engine **257**. Of course, the text can also be displayed for additional feedback purposes on a monitor used with client side system 150. The text to speech engine uses a natural language voice data file **256** associated with it that is appropriate for the particular language application (i.e., English, French, German, Japanese, etc.). As explained earlier when the answer is something more than text, it can be treated as desired to provide responsive information to the user, such as with a graphics image, a sound, a video clip, etc.

Uninitialization

The un-initialization routines and processes are illustrated in FIG. 4. Three functional modules are used for un-initializing the primary components of the client side system 150; these include SRE 270, Communications 271 and MS Agent 272 un-initializing routines. To un-initialize SRE 220A, memory that was allocated in the initialization phase is de-allocated by code 273 and objects created during such initialization phase are deleted by code 274. Similarly, as illustrated in FIG. 4, to un-initialize Communications module 220C the INTERNET connection previously established with the server is closed by code portion 275 of the Communication Un-initialization routine 271. Next the INTERNET session created at the time of initialization is also closed by routine 276. For the un-initialization of the MS Agent 220B, as illustrated in FIG. 4, MS Agent Un-initialization routine 272 first releases the Commands Interface 227 using routine 277. This releases the commands added to the property sheet during loading of the agent character by routine 225. Next the Character Interface initialized by routine 226 is released by routine 278 and the Agent is unloaded at 279. The Sink Object Interface is then also released 280 followed by the release of the Property Sheet Interface 281. The Agent Notify Sink 282 then un-registers the Agent and finally the Agent Interface 283 is released which releases all the resources allocated during initialization steps identified in FIG. 2D.

It will be appreciated by those skilled in the art that the particular implementation for such un-initialization processes and routines in FIG. 4 will vary from client platform to client platform, as for the other routines discussed above. The structure, operation, etc. of such routines are well-known in the art, and they can be implemented using a number of fairly straightforward approaches without undue effort. Accordingly, they are not discussed in detail herein.

DESCRIPTION OF SERVER SIDE SYSTEM 180

Introduction

A high level flow diagram of the set of preferred processes implemented on server side system 180 of Natural Language Query System 100 is illustrated in **Figs. 11A** through **Fig. 11C**. In a preferred embodiment, this process consists of a two step algorithm for completing the processing of the speech input signal, recognizing the meaning of the user's query, and retrieving an appropriate answer/response for such query.

The 1st step as illustrated in **Fig. 11A** can be considered a high-speed first-cut pruning mechanism, and includes the following operations: after completing processing of the speech input signal, the user's query is recognized at step **1101**, so that the text of the query is simultaneously sent to Natural Language Engine 190 (FIG. 1) at step **1107**, and to DB Engine 186 (also FIG.1) at step **1102**. By "recognized" in this context it is meant that the user's query is converted into a text string of distinct native language words through the HMM technique discussed earlier.

At NLE 190, the text string undergoes morphological linguistic processing at step **1108**: the string is tokenized the tags are tagged and the tagged tokens are grouped Next the noun phrases (NP) of the string are stored at **1109**, and also copied and transferred for use by DB Engine 186 during a DB Process at step **1110**. As illustrated in **Fig. 11A**, the string corresponding to the user's query which was sent to the DB Engine 186 at **1102**, is used together with the NP received from NLE 190 to construct an SQL Query at step **1103**. Next, the SQL query is executed at step **1104**, and a record set of potential questions corresponding to the user's query are received as a result of a full-text search at **1105**, which are then sent back to NLE 190 in the form of an array at step **1106**.

As can be seen from the above, this first step on the server side processing acts as an efficient and fast pruning mechanism so that the universe of potential "hits" corresponding to the user's actual query is narrowed down very quickly to a manageable set of likely candidates in a very short period of time.

Referring to **Fig. 11B**, in contrast to the first step above, the 2nd step can be considered as the more precise selection portion of the recognition process. It begins with linguistic processing of each of the stored questions in the array returned by the full-text search process as possible candidates representing the user's query. Processing of these stored questions continues in NLE 190 as follows: each question in the array of questions corresponding to the record set returned by the SQL full-text search undergoes morphological linguistic processing at step **1111**: in this operation, a

text string corresponding to the retrieved candidate question is tokenized, the tags are tagged and the tagged tokens are grouped. Next, noun phrases of the string are computed and stored at step **1112**. This process continues iteratively at point **1113,** and the sequence of steps at **1118,1111, 1112, 1113** are repeated so that an NP for each retrieved candidate question is computed and stored. Once an NP is computed for each of the retrieved candidate questions of the array, a comparison is made between each such retrieved candidate question and the user's query based on the magnitude of the NP value at step **1114**. This process is also iterative in that steps **1114, 1115, 1116, 1119** are repeated so that the comparison of the NP for each retrieved candidate question with that of the NP of the user's query is completed. When there are no more stored questions in the array to be processed at step **1117**, the stored question that has the maximum NP relative to the user's query, is identified at **1117A** as the stored question which best matches the user's query.

Notably, it can be seen that the second step of the recognition process is much more computationally intensive than the first step above, because several text strings are tokenized, and a comparison is made of several NPs. This would not be practical, nonetheless, if it were not for the fact that the first step has already quickly and efficiently reduced the candidates to be evaluated to a significant degree. Thus, this more computationally intensive aspect of the present invention is extremely valuable, however because it yields extremely high accuracy in the overall query recognition process. In this regard, therefore, this second step of the query recognition helps to ensure the overall accuracy of the system, while the first step helps to maintain a satisfactory speed that provides a real-time feel for the user.

As illustrated in **Fig. 11C,** the last part of the query/response process occurs by providing an appropriate matching answer/response to the user. Thus, an identity of a matching stored question is completed at step 1120. Next a file path corresponding to an answer of the identified matching question is extracted at step **1121**. Processing continues so that the answer is extracted from the file path at **1122** and finally the answer is compressed and sent to client side system 150 at step **1123.**

The discussion above is intended to convey a general overview of the primary components, operations, functions and characteristics of those portions of NLQS system 100 that reside on server side system 180. The discussion that follows describes in more detail the respective sub-systems.

Software modules used in Server Side System 180

The key software modules used on server-side system 180 of the NLQS system are

illustrated in **Fig. 5**. These include generally the following components: a Communication module 500 – identified as CommunicationServer ISAPI **500A** (which is executed by SRE Server-side 182 – FIG. 1 and is explained in more detail below), and a database process DBProcess module **501** (executed by DB Engine 186 – FIG. 1). Natural language engine module 500C (executed by NLE 190 – FIG.1) and an interface 500B between the NLE process module 500C and the DBProcess module **500B**. As shown here, CommunicationServerISAPI **500A** includes a server-side speech recognition engine and appropriate communication interfaces required between client side system 150 and server side system 180. As further illustrated in **Fig. 5**, server-side logic of Natural Language Query System 100 also can be characterized as including two dynamic link library components: CommunicationServerISAPI **500** and DBProcess **501**. The CommunicationServerIASPI 500 is comprised of 3 sub-modules: Server-side Speech Recognition Engine module **500A**; Interface module 500B between Natural Language Engine modules 500C and DBProcess 501; and the Natural Language Engine modules **500C.**

DB Process **501** is a module whose primary function is to connect to a SQL database and to execute an SQL query that is composed in response to the user's query. In addition, this module interfaces with logic that fetches the correct answer from a file path once this answer is passed to it from the Natural Language Engine module **500C**.

Speech Recognition Sub-System 182 on Server-Side System 180

The server side speech recognition engine module **500A** is a set of distributed components that perform the necessary functions and operations of speech recognition engine 182 (FIG.1) at server-side 180. These components can be implemented as software routines that are executed by server side 180 in conventional fashion. Referring to **Fig. 4A**, a more detailed break out of the operation of the speech recognition components 600 at the server-side can be seen as follows:

Within a portion **601** of the server side SRE module 500A, the binary MFCC vector byte stream corresponding to the speech signal's acoustic features extracted at client side system 150 and sent over the communication channel 160 is received. The MFCC acoustic vectors are decoded from the encoded HTTP byte stream as follows: Since the MFCC vectors contain embedded NULL characters, they cannot be transferred in this form to server side system 180 as such using HTTP protocol. Thus the MFCC vectors are first encoded at client-side 150 before transmission in such a way that all the speech data is converted into a stream of bytes without embedded NULL characters in the data. At the very end of the byte stream a single NULL character is introduced to indicate the

termination of the stream of bytes to be transferred to the server over the INTERNET 160A using HTTP protocol.

As explained earlier, to conserve latency time between the client and server, a smaller number of bytes (just the 13 MFCC coefficients) are sent from client side system 150 to server side system 180. This is done automatically for each platform to ensure uniformity, or can be tailored by the particular application environment – i.e., such as where it is determined that it will take less time to compute the delta and acceleration coefficients at the server (26 more calculations), than it would take to encode them at the client, transmit them, and then decode them from the HTTP stream. In general, since server side system 180 is usually better equipped to calculate the MFCC delta and acceleration parameters, this is a preferable choice. Furthermore, there is generally more control over server resources compared to the client's resources, which means that future upgrades, optimizations, etc., can be disseminated and shared by all to make overall system performance more reliable and predictable. So, the present invention can accommodate even the worst-case scenario where the client's machine may be quite thin and may just have enough resources to capture the speech input data and do minimal processing.

Dictionary Preparation & Grammar Files

Referring to **Fig. 4A**, within code block **605**, various options selected by the user (or gleaned from the user's status within a particular application) are received. For instance, in the case of a preferred remote learning system, Course, Chapter and/or Section data items are communicated. In the case of other applications (such as e-commerce) other data options are communicated, such as the Product Class, Product Category, Product Brand, etc. loaded for viewing within his/her browser. These selected options are based on the context experienced by the user during an interactive process, and thus help to limit and define the scope – i.e. grammars and dictionaries that will be dynamically loaded to speech recognition engine 182 (FIG. 1) for Viterbi decoding during processing of the user speech utterance. For speech recognition to be optimized both grammar and dictionary files are used in a preferred embodiment. A Grammar file supplies the universe of available user queries; i.e., all the possible words that are to be recognized. The Dictionary file provides phonemes (the information of how a word is pronounced – this depends on the specific native language files that are installed – for example, UK English or US English) of each word contained in the grammar file. It is apparent that if all the sentences for a given environment that can be recognized were contained in a single grammar file then recognition accuracy would be

deteriorated and the loading time alone for such grammar and dictionary files would impair the speed of the speech recognition process.

To avoid these problems, specific grammars are dynamically loaded or actively configured as the current grammar according to the user's context, i.e., as in the case of a remote learning system, the Course, Chapter and/or Section selected. Thus the grammar and dictionary files are loaded dynamically according to the given Course, Chapter and/or Section as dictated by the user, or as determined automatically by an application program executed by the user.

The second code block 602 implements the initialization of Speech Recognition engine 182 (FIG.1). The MFCC vectors received from client side system 150 along with the grammar filename and the dictionary file names are introduced to this block to initialize the speech decoder.

As illustrated in Fig. 4A, the initialization process 602 uses the following sub-routines: A routine 602a for loading an SRE library. This then allows the creation of an object identified as External Source with code 602b using the received MFCC vectors. Code 602c allocates memory to hold the recognition objects. Routine 602d then also creates and initializes objects that are required for the recognition such as: Source, Coder, Recognizer and Results Loading of the Dictionary created by code 602e, Hidden Markov Models (HMMs) generated with code 602f; and Loading of the Grammar file generated by routine 602g.

Speech Recognition **603** is the next routine invoked as illustrated in **Fig. 4A**, and is generally responsible for completing the processing of the user speech signals input on the client side 150, which, as mentioned above, are preferably only partially processed (i.e., only MFCC vectors are computed during the first phase) when they are transmitted across link 160. Using the functions created in External Source by subroutine 602b, this code reads MFCC vectors, one at a time from an External Source **603a**, and processes them in block **603b** to realize the words in the speech pattern that are symbolized by the MFCC vectors captured at the client. During this second phase, an additional 13 delta coefficients and an additional 13 acceleration coefficients are computed as part of the recognition process to obtain a total of 39 observation vectors $\mathbf{O}_t$ referred to earlier. Then, using a set of previously defined Hidden Markov Models (HMMs), the words corresponding to the user's speech utterance are determined in the manner described earlier. This completes the word "recognition" aspect of the query processing, which results are used further below to complete the query processing operations.

It will be appreciated by those skilled in the art that the distributed nature and rapid performance of the word recognition process, by itself, is extremely useful and may be implemented

in connection with other environments that do not implicate or require additional query processing operations. For example, some applications may simply use individual recognized words for filling in data items on a computer generated form, and the aforementioned systems and processes can provide a rapid, reliable mechanism for doing so.

Once the user's speech is recognized, the flow of SRE 182 passes to Un-initialize SRE routine 604 where the speech engine is un-initialized as illustrated. In this block all the objects created in the initialization block are deleted by routine 604a, and memory allocated in the initialization block during the initialization phase are removed by routine 604b.

Again, it should be emphasized that the above are merely illustrative of embodiments for implementing the particular routines used on a server side speech recognition system of the present invention. Other variations of the same that achieve the desired functionality and objectives of the present invention will be apparent from the present teachings.

Database Processor 186 Operation – DBProcess

Construction of an SQL Query used as part of the user query processing is illustrated in **Fig. 4B,** a SELECT SQL statement is preferably constructed using a conventional **CONTAINS** predicate. Module **950** constructs the SQL query based on this SELECT SQL statement, which query is used for retrieving the best suitable question stored in the database corresponding to the user's articulated query, (designated as Question here). A routine 951 then concatenates a table name with the constructed **SELECT** statement. Next, the number of words present in each Noun Phrase of Question asked by the user is calculated by routine **952**. Then memory is allocated by routine **953** as needed to accommodate all the words present in the NP. Next a word List (identifying all the distinct words present in the NP) is obtained by routine 954. After this, this set of distinct words are concatenated by routine 955 to the SQL Query separated with a **NEAR ( )** keyword. Next, the **AND** keyword is concatenated to the SQL Query by routine 956 after each NP. Finally memory resources are freed by code 957 so as to allocate memory to store the words received from NP for any next iteration. Thus, at the end of this process, a completed SQL Query corresponding to the user's articulated question is generated.

Connection to SQL Server – As illustrated in **Fig. 4C**, after the SQL Query is constructed by routine 710, a routine **711** implements a connection to the query database 717 to continue processing of the user query. The connection sequence and the subsequent retrieved record set is

implemented using routines 700 which include the following:

1. Server and database names are assigned by routine 711A to a DBProcess member variable
2. A connection string is established by routine 711B;
3. The SQL Server database is connected under control of code **711C**
4. The SQL Query is received by routine **712A**
5. The SQL Query is executed by code **712B**
6. Extract the total number of records retrieved by the query – **713**
7. Allocate the memory to store the total number of paired questions – **713**
8. Store the entire number of paired questions into an array – **713**

Once the Best Answer ID is received at 716 FIG 4C, from the NLE 14 (FIG 5), the code corresponding 716C receives it passes it to code in 716B where the path of the Answer file is determined using the record number. Then the file is opened 716C using the path passed to it and the contents of the file corresponding to the answer is read. Then the answer is compressed by code in 716D and prepared for transmission over the communication channel 160B (FIG. 1).

NLQS Database 188 - Table Organization

Figure 6 illustrates a preferred embodiment of a logical structure of tables used in a typical NLQS database 188 (FIG.1). When NLQS database 188 is used as part of NLQS query system 100 implemented as a remote learning/training environment, this database will include an organizational multi-level hierarchy that consists typically of a Course **701**, which is made of several chapters **702**, **703**, **704**. Each of these chapters can have one or more Sections **705**, **706**, **707** as shown for Chapter 1. A similar structure can exist for Chapter 2, Chapter 3 ... Chapter N. Each section has a set of one or more question - answer pairs **708** stored in tables described in more detail below. While this is an appropriate and preferable arrangement for a training/learning application, it is apparent that other implementations would be possible and perhaps more suitable for other applications such as e-commerce, e-support, INTERNET browsing, etc., depending on overall system parameters.

It can be seen that the NLQS database 188 organization is intricately linked to the switched grammar architecture described earlier. In other words, the context (or environment) experienced by the user can be determined at any moment in time based at the selection made at the section level, so that only a limited subset of question-answer pairs 708 for example are appropriate for section 705. This in turn means that only a particular appropriate grammar for such question-answer pairs may be switched in for handling user queries while the user is experiencing such

context. In a similar fashion, an e-commerce application for an INTERNET based business may consist of a hierarchy that includes a first level "home" page 701 identifying user selectable options (product types, services, contact information, etc.), a second level may include one or more "product types" pages 702, 703, 704, a third page may include particular product models 705, 706, 707, etc., and with appropriate question-answer pairs 708 and grammars customized for handling queries for such product models. Again, the particular implementation will vary from application to application, depending on the needs and desires of such business, and a typical amount of routine optimization will be necessary for each such application.

Table Organization

In a preferred embodiment, an independent database is used for each Course. Each database in turn can include three types of tables as follows: a Master Table as illustrated in **Figure 7A**, at least one Chapter Table as illustrated in **Figure 7B** and at least one Section Table as illustrated in **Figure 7C**.

As illustrated in **Fig. 7A,** a preferred embodiment of a Master Table has six columns – Field Name **701A**, Data Type **702A**, Size **703A**, Null **704A**, Primary Key **705A** and Indexed **706A**. These parameters are well-known in the art of database design and structure. The Master Table has only two fields – Chapter Name **707A** and Section Name **708A**. Both ChapterName and Section Name are commonly indexed.

A preferred embodiment of a Chapter Table is illustrated in **Figure 7B**. As with the Master Table, the Chapter Table has six (6) columns – Field Name **720**, Data Type **721**, Size **722**, Null **723**, Primary Key **724** and Indexed **725**. There are nine (9) rows of data however, in this case, – Chapter_ID **726**, Answer_ID **727**, Section Name **728**, Answer_Title **729**, PairedQuestion **730**, AnswerPath **731**, Creator **732**, Date of Creation **733** and Date of Modification **734**.

An explanation of the Chapter Table fields is provided in **Figure 7C**. Each of the eight (8) Fields **720** has a description 735 and stores data corresponding to:

AnswerID **727** – an integer that is automatically incremented for each answer given for user convenience

Section_Name **728** – the name of the section to which the particular record belongs. This field along with the AnswerID is used as the primary key

Answer_Title **729** – A short description of the title of the answer to the user query

PairedQuestion **730** – Contains one or more combinations of questions for the related

answers whose path is stored in the next column AnswerPath

AnswerPath **731** – contains the path of a file, which contains the answer to the related questions stored in the previous column; in the case of a pure question/answer application, this file is a text file, but, as mentioned above, could be a multi-media file of any kind transportable over the data link 160

Creator **732**– Name of Content Creator

Date_of_Creation **733** – Date on which content was created

Date of Modification **734** – Date on which content was changed or modified

A preferred embodiment of a Section Table is illustrated in **Figure 7D**. The Section Table has six (6) columns – Field Name **740**, Data Type **741**, Size **742**, Null **743**, Primary Key **744** and Indexed **745**. There are seven (7) rows of data – Answer_ID **746**, Answer_Title **747**, PairedQuestion **748**, AnswerPath **749**, Creator **750**, Date of Creation **751** and Date of Modification **752**. These names correspond to the same fields, columns already described above for the Master Table and Chapter Table.

Again, this is a preferred approach for the specific type of learning/training application described herein. Since the number of potential applications for the present invention is quite large, and each application can be customized, it is expected that other applications (including other learning/training applications) will require and/or be better accommodated by another table, column, and field structure/hierarchy.

Search Service and Search Engine – A query text search service is performed by an SQL Search System 1000 shown in FIG. 10. This system provides querying support to process full-text searches. This is where full-text indexes reside.

In general, SQL Search System determines which entries in a database index meet selection criteria specified by a particular text query that is constructed in accordance with an articulated user speech utterance. The Index Engine 1011B is the entity that populates the Full-Text Index tables with indexes which correspond to the indexable units of text for the stored questions and corresponding answers. It scans through character strings, determines word boundaries, removes all noise words and then populates the full-text index with the remaining words. For each entry in the full text database that meets the selection criteria, a unique key column value and a ranking value are returned as well. Catalog set 1013 is a file-system directory that is accessible only by an Administrator and Search Service 1010. Full-text indexes 1014 are organized into full-text catalogs, which are referenced by easy to handle names. Typically, full-text index data for an entire database is

placed into a single full-text catalog.

The schema for the full-text database as described (FIG 7, Fig 7A, FIG 7B, FIG 7C, FIG 7D) is stored in the tables 1006 shown in FIG. 10. Take for example, the tables required to describe the structure the stored question/answer pairs required for a particular course. For each table – Course Table, Chapter Table, Section Table, there are fields – column information that define each parameters that make up the logical structure of the table. This information is stored in User and System tables 1006. The key values corresponding to those tables are stored as Full-Text catalogs 1013. So when processing a full-text search, the search engine returns to the SQL Server the key values of the rows that match the search criteria. The relational engine then uses this information to respond to the query.

As illustrated in **Fig 10**, a Full-Text Query Process is implemented as follows:

1. A query **1001** that uses a SQL full-text construct generated by DB processor 186 *is* submitted to SQL Relational Engine 1002.
2. Queries containing either a **CONTAINS** or **FREETEXT** predicate are rewritten by routine **1003** so that a responsive rowset returned later from Full-Text Provider 1007 will be automatically joined to the table that the predicate is acting upon. This rewrite is a mechanism used to ensure that these predicates are a seamless extension to a traditional SQL Server. After the compiled query is internally rewritten and checked for correctness in item 1003, the query is passed to RUN TIME module 1004. The function of module 1004 is to convert the rewritten SQL construct to a validated run-time process before it is sent to the Full-Text Provider, 1007.
3. After this, Full-Text Provider 1007 is invoked, passing the following information for the query:
    a. A *ft_search_condition* parameter (this is a logical flag indicating a full text search condition)
    b. A name of a full-text catalog where a full-text index of a table resides
    c. A locale ID to be used for language (for example, word breaking)
    d. Identities of a database, table, and column to be used in the query
    e. If the query is comprised of more than one full-text construct; when this is the case Full-text provider 1007 is invoked separately for each construct.
4. SQL Relational Engine **1002** does not examine the contents of *ft_search_condition.* Instead, this information is passed along to Full-text provider **1007**, which verifies

the validity of the query and then creates an appropriate internal representation of the full-text search condition.

5. The query request/command **1008** is then passed to Querying Support **1011A**.
6. Querying Support **1012** returns a rowset **1009** from Full-Text Catalog **1013** that contains unique key column values for any rows that match the full-text search criteria. A rank value also is returned for each row.
7. The rowset of key column values 1009 is passed to SQL Relational Engine **1002.** If processing of the query implicates either a **CONTAINSTABLE()** or **FREETEXTTABLE()** function, **RANK** values are returned; otherwise, any rank value is filtered out.
8. The rowset values 1009 are plugged into the initial query with values obtained from relational database 1006, and a result set **1015** is then returned for further processing to yield a response to the user.

At this stage of the query recognition process, the speech utterance by the user has already been rapidly converted into a carefully crafted text query, and this text query has been initially processed so that an initial matching set of results can be further evaluated for a final determination of the appropriate matching question/answer pair. The underlying principle that makes this possible is the presence of a full-text unique key column for each table that is registered for full-text searches. Thus when processing a full-text search, SQL Search Service 1010 returns to SQL server 1002 the key values of the rows that match the database. In maintaining these full-text databases 1013 and full text indexes 1014, the present invention has the unique characteristic that the full-text indices 1014 are not updated instantly when the full-text registered columns are updated. This operation is eliminated, again, to reduce recognition latency, increase response speed, etc. Thus, as compared to other database architectures, this updating of the full-text index tables, which would otherwise take a significant time, is instead done asynchronously at a more convenient time.

Interface between NLE 190 and DB Processor 188

The result set 1015 of candidate questions corresponding to the user query utterance are presented to NLE 190 for further processing as shown in FIG. 4D to determine a "best" matching question/answer pair. An NLE/DBProcessor interface module coordinates the handling of user queries, analysis of noun-phrases (NPs) of retrieved questions sets from the SQL query based on the user query, comparing the retrieved question NPs with the user query NP, etc. between NLE 190

and DB Processor 188. So, this part of the server side code contains functions, which interface processes resident in both NLE block 190 and DB Processor block 188. The functions are illustrated in **Fig. 4D;** As seen here, code routine **880** implements functions to extract the Noun Phrase (NP) list from the user's question. This part of the code interacts with NLE 190 and gets the list of Noun Phrases in a sentence articulated by the user. Similarly, Routine **813** retrieves an NP list from the list of corresponding candidate/paired questions 1015 and stores these questions into an (ranked by NP value) array. Thus, at this point, NP data has been generated for the user query, as well as for the candidate questions 1015. As an example of determining the noun phrases of a sentence such as: "*What issues have guided the President in considering the impact of foreign trade policy on American businesses*?" NLE 190 would return the following as noun phrases: *President, issues, impact of foreign trade policy, American businesses, impact, impact of foreign trade, foreign trade, foreign trade policy, trade, trade policy, policy, businesses.* The methodology used by NLE 190 will thus be apparent to those skilled in the art from this set of noun phrases and noun sub-phrases generated in response to the example query.

Next, a function identified as Get Best Answer ID 815 is implemented. This part of the code gets a best answer ID corresponding to the user's query. To do this, routines 813A, 813B first find out the number of Noun phrases for each entry in the retrieved set 1015 that match with the Noun phrases in the user's query. Then routine 815a selects a final result record from the candidate retrieved set 1015 that contains the maximum number of matching Noun phrases.

Conventionally, nouns are commonly thought of as "naming" words, and specifically as the names of "people, places, or things". Nouns such as *John*, *London*, and *computer* certainly fit this description, but the types of words classified by the present invention as nouns is much broader than this. Nouns can also denote abstract and intangible concepts such as *birth*, *happiness*, *evolution*, *technology*, *management*, *imagination*, *revenge*, *politics*, *hope*, *cookery*, *sport*, *and literacy*. Because of the enormous diversity of nouns compared to other parts of speech, the Applicant has found that it is much more relevant to consider the noun phrase as a key linguistic metric. So, the great variety of items classified as nouns by the present invention helps to discriminate and identify individual speech utterances much easier and faster than prior techniques disclosed in the art.

Following this same thought, the present invention also adopts and implements another linguistic entity – the word phrase - to facilitate speech query recognition. The basic structure of a word phrase – whether it be a noun phrase, verb phrase, adjective phrase - is three parts – [pre-Head string],[Head] and [post-Head string]. For example, in the minimal noun phrase - "the children,"

"children" is classified as the Head of the noun phrase. In summary, because of the diversity and frequency of noun phrases, the choice of noun phrase as the metric by which stored answer is linguistically chosen, has a solid justification in applying this technique to the English natural language as well as other natural languages. So, in sum, the total noun phrases in a speech utterance taken together operate extremely well as unique type of speech query fingerprint.

The ID corresponding to the best answer corresponding to the selected final result record question is then generated by routine 815 which then returns it to DB Process shown in FIG.4C. As seen there, a Best Answer ID I is received by routine 716A, and used by a routine 716B to retrieve an answer file path. Routine 716C then opens and reads the answer file, and communicates the substance of the same to routine 716D. The latter then compresses the answer file data, and sends it over data link 160 to client side system 150 for processing as noted earlier (i.e., to be rendered into audible feedback, visual text/graphics, etc.). Again, in the context of a learning/instructional application, the answer file may consist solely of a single text phrase, but in other applications the substance and format will be tailored to a specific question in an appropriate fashion. For instance, an "answer" may consist of a list of multiple entries corresponding to a list of responsive category items (i.e., a list of books to a particular author) etc. Other variations will be apparent depending on the particular environment.

Natural Language Engine 190

Again referring to **Fig. 4D**, the general structure of NL engine 190 is depicted. This engine implements the word analysis or morphological analysis of words that make up the user's query, as well as phrase analysis of phrases extracted from the query.

As illustrated in **Fig. 9**, the functions used in a morphological analysis include tokenizers **802A**, stemmers **804A** and morphological analyzers **806A**. The functions that comprise the phrase analysis include tokenizers, taggers and groupers, and their relationship is shown in FIG.8.

Tokenizer 802A is a software module that functions to break up text of an input sentence 801A into a list of tokens **803A**. In performing this function, tokenizer 802A goes through input text 801A and treats it as a series of tokens or useful meaningful units that are typically larger than individual characters, but smaller than phrases and sentences. These tokens 803A can include words, separable parts of word and punctuation. Each token 803A is given an offset and a length. The first phase of tokenization is segmentation, which extracts the individual tokens from the input text and keeps track of the offset where each token originated from in the input text. Next, categories are

associated with each token, based on its shape. The process of tokenization is well-known in the art, so it can be performed by any convenient application suitable for the present invention.

Following tokenization, a stemmer process 804A is executed, which can include two separate forms – inflectional and derivational, for analyzing the tokens to determine their respective stems 805A. An inflectional stemmer recognizes affixes and returns the word which is the stem. A derivational stemmer on the other hand recognizes derivational affixes and returns the root word or words. While stemmer 804A associates an input word with its stem, it does not have parts of speech information. Analyzer 806B takes a word independent of context, and returns a set of possible parts of speech 806A.

As illustrated in **Fig. 8**, phrase analysis 800 is the next step that is performed after tokenization. A tokenizer 802 generates tokens from input text 801. Tokens 803 are assigned to parts of a speech tag by a tagger routine **804**, and a grouper routine **806** recognizes groups of words as phrases of a certain syntactic type. These syntactic types include for example the noun phrases mentioned earlier, but could include other types if desired such as verb phrases and adjective phrases. Specifically, tagger 804 is a parts-of-speech disambiguator, which analyzes words in context. It has a built-in morphological analyzer (not shown) that allows it to identify all possible parts of speech for each token. The output of tagger 804 is a string with each token tagged with a parts-of-speech label 805. The final step in the linguistic process 800 is the grouping of words to form phrases 807. This function is performed by the grouper **806**, and is very dependent, of course, on the performance and output of tagger component 804.

Accordingly, at the end of linguistic processing 800, a list of noun phrases (NP) 807 is generated in accordance with the user's query utterance. This set of NPs generated by NLE 190 helps significantly to refine the search for the best answer, so that a single-best answer can be later provided for the user's question.

The particular components of NLE **190** are shown in FIG. 4D, and include several components. Each of these components implement the several different functions required in NLE 190 as now explained.

Initialize Grouper Resources Object and the Library **900** – this routine initializes the structure variables required to create grouper resource object and library. Specifically, it initializes a particular natural language used by NLE 190 to create a Noun Phrase, for example the English natural language is initialized for a system that serves the English language market. In turn, it also creates the objects (routines) required for Tokenizer, Tagger and Grouper (discussed above) with

routines 900A, 900B, 900C and 900D respectively, and initializes these objects with appropriate values. It also allocates memory to store all the recognized Noun Phrases for the retrieved question pairs.

Tokenizing of the words from the given text (from the query or the paired questions) is performed with routine **909B** – here all the words are tokenized with the help of a local dictionary used by NLE 190 resources. The resultant tokenized words are passed to a Tagger routine 909C. At routine 909C, tagging of all the tokens is done and the output is passed to a Grouper routine 909D.

The Grouping of all tagged token to form NP list is implemented by routine 909D so that the Grouper groups all the tagged token words and outputs the Noun Phrases.

Un-initializing of the grouper resources object and freeing of the resources, is performed by routines 909EA, 909EB and 909EC. These include Token Resources, Tagger Resources and Grouper Resources respectively. After initialization, the resources are freed. The memory that was used to store all Noun Phrases are also de-allocated.

Additional Embodiments

In a e-commerce embodiment of the present invention as illustrated in FIG. 13, a web page 1300 contains typical visible links such as Books 1310, Music 1320 so that on clicking the appropriate link the customer is taken to those pages. The web page may be implemented using HTML, a Java applet, or similar coding techniques which interact with the user's browser. For example, if customer wants to buy an album C by Artist Albert, he traverses several web pages as follows: he first clicks on Music (FIG. 13, 1360), which brings up page 1400 where he/she then clicks on Records (FIG. 14, 1450). Alternatively, he/she could select CDs 1460, Videos 1470, or other categories of books 1410, music 1420 or help 1430. As illustrated in FIG. 15, this brings up another web page 1500 with links for Records 1550, with sub-categories – Artist 1560, Song 1570, Title 1580, Genre 1590. The customer must then click on Artist 1560 to select the artist of choice. This displays another web page 1600 as illustrated in FIG. 16. On this page the various artists 1650 are listed as illustrated – Albert 1650, Brooks 1660, Charlie 1670, Whyte 1690 are listed under the category Artists 1650. The customer must now click on Albert 1660 to view the albums available for Albert. When this is done, another web page is displayed as shown in FIG. 17. Again this web page 1700 displays a similar look and feel, but with the albums available 1760, 1770, 1780 listed under the heading Titles 1750. The customer can also read additional information 1790 for each album. This album information is similar to the liner notes of a shrink-wrapped album purchased at a retail store.

One Album A is identified, the customer must click on the Album A 1760. This typically brings up another text box with the information about its availability, price, shipping and handling charges etc.

When web page 1300 is provided with functionality of a NLQS of the type described above, the web page interacts with the client side and server side speech recognition modules described above. In this case, the user initiates an inquiry by simply clicking on a button designated Contact Me for Help 1480 (this can be a link button on the screen, or a key on the keyboard for example) and is then told by character 1440 about how to elicit the information required. If the user wants Album A by artist Albert, the user could articulate "Is Album A by Brooks available?" in much the same way they would ask the question of a human clerk at a brick and mortar facility. Because of the rapid recognition performance of the present invention, the user's query would be answered in real-time by character 1440 speaking out the answer in the user's native language. If desired, a readable word balloon 1490 could also be displayed to see the character's answer and so that save/print options can also be implemented. Similar appropriate question/answer pairs for each page of the website can be constructed in accordance with the present teachings, so that the customer is provided with an environment that emulates a normal conversational human-like question and answer dialog for all aspects of the web site. Character 1440 can be adjusted and tailored according to the particular commercial application, or by the user's own preferences, etc. to have a particular voice style (man, woman, young, old, etc.) to enhance the customer's experience.

In a similar fashion, an articulated user query might be received as part of a conventional search engine query, to locate information of interest on the INTERNET in a similar manner as done with conventional text queries. If a reasonably close question/answer pair is not available at the server side (for instance, if it does not reach a certain confidence level as an appropriate match to the user's question) the user could be presented with the option of increasing the scope so that the query would then be presented simultaneously to one or more different NLEs across a number of servers, to improve the likelihood of finding an appropriate matching question/answer pair. Furthermore, if desired, more than one "match" could be found, in the same fashion that conventional search engines can return a number of potential "hits" corresponding to the user's query. For some such queries, of course, it is likely that real-time performance will not be possible (because of the disseminated and distributed processing) but the advantage presented by extensive supplemental question/answer database systems may be desirable for some users.

It is apparent as well that the NLQS of the present invention is very natural and saves much time for the user and the e-commerce operator as well. In an e-support embodiment, the customer

can retrieve information quickly and efficiently, and without need for a live customer agent. For example, at a consumer computer system vendor related support site, a simple diagnostic page might be presented for the user, along with a visible support character to assist him/her. The user could then select items from a "symptoms" page (i.e., a "monitor" problem, a "keyboard" problem, a "printer" problem, etc.) simply by articulating such symptoms in response to prompting from the support character. Thereafter, the system will direct the user on a real-time basis to more specific sub-menus, potential solutions, etc. for the particular recognized complaint. The use of a programmable character thus allows the web site to be scaled to accommodate a large number of hits or customers without any corresponding need to increase the number of human resources and its attendant training issues.

As an additional embodiment, the searching for information on a particular web site may be accelerated with the use of the NLQS of the present invention. Additionally, a significant benefit is that the information is provided in a user-friendly manner through the natural interface of speech. The majority of web sites presently employ lists of frequently asked questions which the user typically wades item by item in order to obtain an answer to a question or issue. For example, as displayed in FIG. 13, the customer clicks on Help 1330 to initiate the interface with a set of lists. Other options include computer related items at 1370 and frequently asked questions (FAQ) at 1380.

As illustrated in FIG. 18, a web site plan for typical web page is displayed. This illustrates the number of pages that have to be traversed in order to reach the list of Frequently-Asked Questions. Once at this page, the user has to scroll and manually identify the question that matches his/her query. This process is typically a laborious task and may or may not yield the information that answers the user's query. The present art for displaying this information is illustrated in FIG. 18. This figure identifies how the information on a typical web site is organized: the Help link (FIG. 13, 1330) typically shown on the home page of the web page is illustrated shown on FIG. 18 as 1800. Again referring to FIG. 18, each sub-category of information is listed on a separate page. For example, 1810 lists sub-topics such as 'First Time Visitors', 'Search Tips', 'Ordering', 'Shipping', 'Your Account' etc. Other pages deal with 'Account information' 1860, 'Rates and Policies' 1850 etc. Down another level, there are pages that deal exclusively with a sub-sub topics on a specific page such as 'First Time Visitors' 1960, 'Frequently Asked Questions' 1950, 'Safe Shopping Guarantee' 1940, etc. So if a customer has a query that is best answered by going to the Frequently Asked Questions link, he or she has to traverse three levels of busy and cluttered screen pages to get

to the Frequently Asked Questions page 1950. Typically, there are many lists of questions 1980 that have to be manually scrolled through. While scrolling visually, the customer then has to visually and mentally match his or her question with each listed question. If a possible match is sighted, then that question is clicked and the answer then appears in text form which then is read.

In contrast, the process of obtaining an answer to a question using a web page enabled with the present NLQS can be achieved much less laboriously and efficiently. The user would articulate the word "Help" (FIG. 13, 1330). This would immediately cause a character (FIG. 13, 1340) to appear with the friendly response "May I be of assistance. Please state your question?". Once the customer states the question, the character would then perform an animation or reply "Thank you, I will be back with the answer soon". After a short period time (preferably not exceeding 5-7 seconds) the character would then speak out the answer to the user's question. As illustrated in FIG. 18 the answer would be the answer 1990 returned to the user in the form of speech is the answer that is paired with the question 1950. For example, the answer 1990: "We accept Visa, MasterCard and Discover credit cards", would be the response to the query 2000 "What forms of payments do you accept?"

Another embodiment of the invention is illustrated in FIG. 12. This web page illustrates a typical website that employs NLQS in a web-based learning environment. As illustrated in FIG. 12, the web page in browser 1200, is divided into two or more frames. A character 1210 in the likeness of an instructor is available on the screen and appears when the student initiates the query mode either by speaking the word "Help" into a microphone (FIG. 2B, 215) or by clicking on the link 'Click to Speak' (FIG. 12, 1280). Character 1210 would then prompt the student to select a course 1220 from the drop down list 1230. If the user selects the course 'CPlusPlus', the character would then confirm verbally that the course "CPlusPlus" was selected. The character would then direct the student to make the next selection from the drop-down list 1250 that contains the selections for the chapters 1240 from which questions are available. Again, after the student makes the selection, the character 1210 confirms the selection by speaking. Next character 1210 prompts the student to select 'Section' 1260 of the chapter from which questions are available from the drop down list 1270. Again, after the student makes the selection, character 1210 confirms the selection by articulating the 'Section' 1260 chosen. As a prompt to the student, a list of possible questions appear in the list box 1291. In addition, tips 1290 for using the system are displayed. Once the selections are all made, the student is prompted by the character to ask the question as follows: "Please ask your query now". The student then speaks his query and after a short period of time, the character

responds with the answer preceded by the question as follows: "The answer to your question ….. is as follows: ……..". This procedure allows the student to quickly retrieve answers to questions about any section of the course and replaces the tedium of consulting books, and references or indices. In short, it is can serve a number of uses from being a virtual teacher answering questions on-the-fly or a flash card substitute.

From preliminary data available to the inventors, it is estimate that the system can easily accommodate 100 – 250 question/answer pairs while still achieving a real-time feel and appearance to the user (i.e., less than 10 seconds of latency, not counting transmission) using the above described structures and methods. It is expected, of course, that these figures will improve as additional processing speed becomes available, and routine optimizations are employed to the various components noted for each particular environment.

Semantic Decoding

In addition to the semantic checking and validation component noted above in connection with the SQL query, another aspect of the present invention concerns semantic decoding to determine the meaning of a speech utterance. As discussed above, the algorithms of many current natural language processing systems use a statistics-based linguistic algorithm to find the correct matches between the user's question with a stored question to retrieve a single best answer. However, many of such systems do not have the capability to handle user questions that have semantic variations with a given user question. For example, if the stored question is: *'How do I reboot my system'*, and the user's question is: *'What do I do when my computer crashes'*, we could, with the help of a lexical dictionary such as WordNet, establish that there is a semantic relationship between *'computer crash'* and *'rebooting'*. This would then allow us to understand the link between *'computer crash'* and *'rebooting my system'*.

WordNet is the product of a research project at Princeton University that has modeled the lexical knowledge of a native speaker of English. For further information, the following URL can be used (using as www prefix) cogsci.princeton.edu/~wn/. WordNet has also been extended to several other languages, including Spanish, Japanese, German and French. The system has capabilities of both an on-line thesaurus and an on-line dictionary. Information in WordNet is organized as a network of nodes. Each of these word sense nodes is a group of synonyms called synsets. Each sense of a word is mapped to such a sense word – i.e. a synset, - the basic building

block, and all word sense nodes in WordNet are linked by a variety of semantic relationships. The basic semantic relationship in WordNet is synonymy. Although synonomy is a semantic relationship between word forms, the semantic relationship that is most important in organizing nouns is a relation between lexical concepts. This relationship is called hyponymy. For example the noun *robin* is a hyponym (subordinate) of the noun *bird*, or conversely *bird* is a hypernym (*superordinate*) of *robin*. WordNet uses this semantic relationship to organize nouns into a lexical hierarchy.

The input to the WordNet is a word or group of words with a single meaning, e.g.,. "co-operation". The output of the WordNet is a synset (a set of synonym and their meanings). The typical interfaces are:

- **findtheinfo():** Primary search function for WordNet database. Returns formatted search results in text buffer. Used by WordNet interfaces to perform requested search.
- **read_synset():** Reads synset from data file at byte offset passed and return parsed entry in data structure called **parse_synset().**
- **parse_synset():** Reads synset at the current byte offset in file and returns the parsed entry in data structure.
- **getsstype():** Returns synset type code for string passed.
- **GetSynsetForSense(char *sense_key):** returns the synset that contains the word sense *sense_key* and **NULL** in case of error.

Thus, one approach to natural language processing is to use a statistics-based implementation that relies on noun phrases to establish how closely matched the user's question is with the stored question. One way in which such processing can be improved is to expand the algorithm to incorporate the capability to establish semantic relationship between the words.

In the present invention therefore, WordNet-derived metrics are used in parallel with a statistics-based algorithm so as to enhance the accuracy of a NLQS Natural Language processing Engine (NLE). Specifically, a speech utterance is processed as noted above, including with a speech recognizer, to extract the words associated with such utterance. Very briefly, an additional function based on rank and derived from one or more metrics as follows:

First Metric:

1. Compare each word of the user's question in the utterance with each word of the stored question. If u_i is the i^{th} word of the user's question, and f_j is the j^{th} word of the stored

question, then the similarity score s between the two words is $S(u_i, f_j)$ is equal to the **minimum** lexical path distance between the two words being compared [and can be later defined to take into account some other constants related to the environment]

2. So for the entire query we a word by word comparison between the user's question and the stored question is carried out, to compute the following matrix, where the user's question has n words and the stored questions have m words:

$$S(u_i, f_j) = \begin{bmatrix} S(u_1,f_1) & \ldots\ldots & S(u_1,f_m) \\ | & & | \\ | & & | \\ | & & | \\ \lfloor S(u_n,f_1) & \ldots\ldots & S(u_n,f_m) \rfloor \end{bmatrix}$$

Note: Some stored questions may have *m* words, others *m-1*, or *m-2*, or *m+1*, or *m+2* words ... etc. So for the matrices may be different order as we begin to compare the user's question with each stored question.

The first row of the matrix calculates the metric for first word of the user's question with each word of the stored question. The second row would calculate the metric for the second word of the user's question with each word of the stored question. The last row would calculate the metric for the last word of the user's question (u_n) with each word of the stored question ($f_1 \ldots to f_m$).

In this way a matrix of coefficients is created between the user's question and the stored question.

The next step is to reduce this matrix to a single value. We do this by choosing the maximum value of the matrix between the user's question and each stored question. We can also employ averaging over all the words that make up the sentence, and also introduce constants to account for other modalities.

Second Metric:

Find the degree of coverage between the user's question and the stored question. We use WordNet to calculate coverage of the stored questions, which will be measured as a percentage of the stored questions covering as much as possible semantically of the user's question. Thus the coverage can represent the percentage of words in the user's question that is covered by the stored

question.

Final Metric:

All of the above metrics are combined into a single metric or rank that includes also weights or constants to adjust for variables in the environment etc.

The NLQS Semantic Decoder - Description

As alluded to above, one type of natural language processing is a statistical-based algorithm that uses noun phrases and other parts of speech to determine how closely matched the user's question is with the stored question. This process is now extended to incorporate the capability to establish semantic relationship between the words, so that correct answers to variant questions are selected. Put another way, a semantic decoder based on computations using a programmatic lexical dictionary was used to implement this semantic relationship. The following describes how WordNet-derived metrics are used to enhance the accuracy of NLQS statistical algorithms.

FIG. 19 illustrates a preferred method 1995 for computing a semantic match between user-articulated questions and stored semantic variants of the same. Specifically, a function is used based on rank and derived from one or more metrics as follows:

Three metrics – *term frequency*, *coverage* and *semantic similarity* are computed to determine the one and only one paired question of a recordset returned by a SQL search that is closest semantically to the user's query. It will be understood by those skilled in the art that other metrics could be used, and the present invention is not limited in this respect. These are merely the types of metrics that are useful in the present embodiment; other embodiments of the invention may benefit from other well-known or obvious variants.

The first metric – the first metric, term frequency, a long established formulation in information retrieval, uses the cosine vector similarity relationship, and is computed at step 1996. A document is represented by a term vector of the form:

$$R = (t_i, t_j, \ldots\ldots t_p) \qquad (1)$$

where each t_k identifies a content term assigned to a record in a recordset

Similarly, a user query can be represented in vector form as:

$$UQ = (q_a, q_b, \ldots\ldots q_r) \qquad (2)$$

The term vectors of (1) and (2) is obtained by including in each vector all possible content terms allowed in the system and adding term weights. So if w_{dk} represents the weight of term t_k in the record R or user query UQ, the term vectors can be written as:

$$= (t_0, w_{r0}; t_1, w_{r1}; \ldots\ldots t_t, w_{rt}) \qquad (3)$$

and

$$UQ = (uq_0, w_{uq0}; uq_1, w_1; \ldots\ldots uq_t, w_{qt}) \qquad (4)$$

A similarity value between the user query (UQ) and the recordset (R) may be obtained by comparing the corresponding vectors using the product formula:

$$\textbf{Similarity (UQ, R)} = \sum_{k=1}^{t} W_{uqk} \cdot W_{rk}$$

A typical term weight using a vector length normalization factor is:

$$\frac{W_{rk}}{\sqrt{\sum_{\text{vector}} (W_{rt})^2}} \quad \textbf{for the recordset}$$

$$\frac{W_{uqk}}{\sqrt{\sum_{\text{vector}} (W_{uqi})^2}} \quad \textbf{for the user query}$$

Using the cosine vector similarity formula we obtain the metric **T**:

$$\mathbf{T = cos\,(v_{uq}, v_r) = Similarity\,(UQ, R)} = \frac{\sum_{k=1}^{t} W_{uqk} W_{rk}}{\sqrt{\sum_{k=1}^{t} (W_{uqk})^2 \sum_{k=1}^{t} (W_{rk})^2}}$$

Thus, the overall similarity between the user query (UQ) and each of the records of the recordset (R) is quantified by taking into account the frequency of occurrence of the different terms in the UQ and each of the records in the recordset. The weight for each term w_i is related to the frequency of that term in the query or the paired question of the recordset, and is **tidf = n x log (M/n)** where **n** is the number of times a term appears, and **M** is the number of questions in the recordset and user query. TIDF refers to **t**erm **f**requency x **i**nverse of **d**ocument **f**requency, again, a well-known term in the art. This metric does not require any understanding of the text – it only takes into account the number of times that a given term appears in the UQ compared to each of the records.

The second metric – the second metric computed during step 1997 – C -- corresponds to coverage, and is defined as the percentage of the number of terms in the user question that appear in each of the records returned by the SQL search.

The third metric - the third metric computed at step 1998 - W - is a measure of the semantic similarity between the UQ and each of the records of the recordset. For this we use WordNet, a programmatic lexical dictionary to compute the semantic distance between two like parts of speech – e.g. noun of UQ and noun of record, verb of UQ and verb of record, adjective of UQ and adjective of record etc. The semantic distance between the user query and the recordset is defined as follows:

$$\text{Sem}\,(T_{uq}, T_r) = [I(uq,r) + I(r,uq)] / [\text{Abs}[T_{uq}] + \text{Abs}[T_r]]$$

where I(uq,r) and I(r,uq) are values corresponding to the inverse semantic distances computed at a given sense and level of WordNet in both directions.

Finally at step 1999 a composite metric M, is derived from the three metrics – T, C and W as follows:

$$\mathbf{M} = \frac{(\mathbf{tT} + \mathbf{cC} + \mathbf{wW})}{(\mathbf{t} + \mathbf{c} + \mathbf{w})}$$

where t, c and w are weights for the corresponding metrics T, C and W.

A standalone software application was then coded to implement and test the above composite metric, M. A set of several test cases was developed to characterize and analyze the algorithm based on WordNet and NLQS natural language engine and search technologies. Each of the test cases were carefully developed and based on linguistic structures. The idea here is that a recordset exists which simulates the response from a SQL search from a full-text database. In a NLQS algorithm of the type described above, the recordset retrieved consists of a number of records greater than 1 – for example 5 or up to 10 records. These records are questions retrieved from the full-text database that are semantically similar to the UQ. The idea is for the algorithm to semantically analyze the user query with each record using each of the three metrics – term frequency T, coverage C, and semantic distance W to compute the composite metric M. Semantic distance metric employs interfacing with the WordNet lexical dictionary. The algorithm uses the

computed value of the composite metric to select the question in the recordset that best matches the UQ.

For example a user query (UQ) and a recordset of 6 questions returned by the SQL search is shown below. Record #4 is known to be the correct question that has the closest semantic match with the UQ. All other records – 1-3, 5, 6 are semantically further away.

Test Case i	Example
UQ: xxxxxxxxxxxxxxxxxxxxxxx	How tall is the Eiffel Tower?
Recordset:	
1. ppppppppppppppppppp	What is the height of the Eiffel Tower?
2. qqqqqqqqqqqqqqqqqqq	How high is the Eiffel Tower?
3. rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr	The Eiffel Tower stands how high?
4. CCCCCCCCCCCCCC	**The Eiffel Tower is how high?**
5. ttttttttttttttttttttttttttttttttttt	How many stories does the Eiffel Tower have?
6. uuuuuuuuuuuuuuuuuu	How high does the Eiffel Tower stand?

Record #4 is designed manually to be semantically the closest to UQ.

Several test cases are created in which each test case has a different UQ. Each UQ is linked to a recordset that has 1 record with the correct semantically matched question. That matching question will have the closest semantic match to the UQ than all the other 5 questions in the recordset.

The standalone application takes each of the test cases one at a time and results are recorded for retrieval time and recall while varying 2 parameters – number of senses and number of levels. The recall determines the efficiency of the algorithm and the metrics derived.

Following the functional testing using a standalone software application, the integration with a NLQS system is as follows:

Assume that the variant question (bold in example) is the question with the closest semantic match with the user query. Also assume that this question has a paired answer that is the answer to the user query. Then the non-bold variants of the recordset represent all the variant questions that can be asked by the user. These non-bold variants are encoded into the NLQS speech lattice, grammar and dictionaries. The user can then query using any of the variant forms or the original query using speech. The query will be speech recognized and the bold question that is stored in the

database will be retrieved as the question pair. Then the corresponding answer to the bold -question will be delivered via text-to-speech.

A test set of several test cases were then designed for the functional testing of extending the capabilities to accurately retrieve correct answers to user questions which varied semantically with the original user query. As an example of a test case is:

User Query	Where are the next Olympic Games being held?
Semantic Variants	• **Where will the next Olympic Games be held?** • Who will next host the Olympics? • In which city will the next Olympics take place? • Which city will next host the Olympics? • Name the site of the next Olympics.

Other examples will be apparent to those skilled in the art from the present teachings. For each application, a different set of semantic variants tailored to such application can be accommodated to improve an overall query/sentence recognition accuracy.

Populating Speech Lattice with Semantic Variants

A complementary process to the above, of course, is shown in FIG. 20, which illustrates a method for populating a speech lattice with semantic variants. This procedure populates a NLQ system speech recognition lattice with a specified number of variant questions of a given user question possible for the domain. The basic approach taken by the invention is that the capabilities of a NLE to process and correctly understand user questions that are semantically similar to the stored question, will enable a NLQS system to provide accurate answers even for uttered questions that are only semantically related to the stored questions.

To accommodate transcription of speech to text, a typical NLQS distributed speech recognition engine uses a word lattice as a grammar, to provide the complete range of hypotheses, of all word sequences that could be spoken. This word lattice can be derived from a manually-written BNF (Backus Naur Form) or finite state grammar, or it could be in the form of an N-gram grammar or statistical language model (LM) which allows all logically possible (even linguistically impossible) word sequences and which reduces the task perplexity via probabilistic modeling of the

N-gram sequences, so that the less likely sequences (observed less frequently or never in a large training dataset) are discarded earlier in the recognizer's search procedure. Thus, the focus of this aspect of the invention is to generate a speech grammar that includes all possible paraphrasings of the questions that an NLQS query system knows how to answer.

The approach taken in the present invention is to generate a smaller N-gram language model, by partitioning a larger N-gram grammar into subsets using the words (and phrases) in our question list along with their synonyms and along with closed-class, grammatical function words that could occur anywhere though they may not happen to be in our target question list. The approach is automatic and statistical rather than intuition-based manual development of linguistic grammars. Given a large N-gram language model (say for simplicity's sake a bigram), the intention is to extract out of the N-gram that subset which will cover the task domain.

Starting with the set of target questions for which we aim to model all the paraphrases, we use a lexical database (such as WordNet or a similarly capable database) to find a set of synonyms and near-synonyms for each content word in each question. Phrasal synonyms could also be considered, perhaps in a second phase. The vocabulary, then, which our sub-setted N-gram language model needs to cover, comprises this full set of target content words and their semantic close cousins, along with the entire inventory of closed-class words of the language (grammar function words which might well occur in any not-known-in-advance sentence).

Next, observation counts underlying a pre-existing N-gram language model (LM), which is much larger and more inclusive (having been trained on a large task-unconstrained dataset), can be copied into a sub-setted statistical language model for those N-grams where each of the N words are contained in the task vocabulary. Probabilities can then be re-estimated (or re-normalized) using these counts, considering that row totals are much reduced in the sub-setted LM as compared with the full LM.

The result is an N-gram statistical language model that should cover the task domain. As usage accumulates and experience grows, it is possible to make additions to the vocabulary and adjustments to the N-gram probabilities based on actual observed task-based data.

In summary, the above procedure is implemented as a tool – called a data preparation tool (DPT) for example. Its function (much like a present NLQS data population tool that is now used to populate the full-text database with question-answer pairs), would be to implement the steps of FIG. 20 to create a speech grammar lattice that would allow recognition of the semantic variants of the user's question.

Therefore, as shown in FIG. 20, the basic steps of the semantic variant question population process 2000 include:

1. Inputting user questions at step 2010 (UQ).
2. Parsing the input question into words or parts of speech at step 2020;
3. Obtaining the synonyms for the parsed words at step 2030;.
4. Using the synonym words to prepare a set of random questions at step 2040;.
5. Verifying and obtaining only the disambiguated set of questions from the random questions at step 2050 using the WordNet semantic decoding (WSD) methodology above.
6. Creating the speech recognition lattice file at step 2060 using the disambiguated set of questions. This lattice file is then used to populate the NLQS Speech Recognition lattice.

In summary the above steps outline a procedure implemented as a software application and used as an adjunct to the semantics-based NLQS natural language engine (NLE) to provide variant questions for any single user question.

Integration of Semantic Algorithm with a statistics-based NLQS algorithm

The semantic algorithm discussed above is easily integrated and implemented along with a NLQS algorithm described above. The integrated algorithm, which can be thought of as a hybrid statistical-semantic language decoder, is shown in Fig. 21. Entry points to the WordNet-based semantic component of the processing are steps 3b and 7.

While the preferred embodiment is directed specifically to integrating the semantic decoder with embodiments of a NLQ system of the type noted above, it will be understood that it could be incorporated within a variety of statistical based NLQ systems. Furthermore, the present invention can be used in both shallow and deep type semantic processing systems of the type noted above.

Appendix: Key WordNet API functions used in the programmatic interface to NLQS Algorithm

The key application programming interfaces which can be used by a preferred embodiment with WordNet are:

1. wninit()

Explanation:

Top level function to open database files and morphology exception lists.

2. is_defined()

Explanation:

Sets a bit for each search type that is valid for *searchstr* in *pos*, and returns the resulting unsigned long integer. Each bit number corresponds to a pointer type constant defined in **WNHOME/include/wnconsts.h** .

3. findtheinfo_ds New

Explanation:

findtheinfo_ds returns a linked list data structures representing synsets. Senses are linked through the *nextss* field of a **Synset** data structure. For each sense, synsets that match the search specified with *ptr_type* are linked through the *ptrlist* field.

findtheinfo_ds is modified into the *findtheinfo_ds_new* function. The modified function will restrict the retrieval of synonyms by searching the wordNet with limited number of senses and traverses limited number of levels.

Explanation:

4. traceptrs_ds New

Explanation:

traceptrs_ds is a recursive search algorithm that traces pointers matching *ptr_type* starting with the synset pointed to by *synptr* . Setting *depth* to **1** when **traceptrs_ds()** is called indicates a recursive search; **0** indicates a non-recursive call. *synptr* points to the data structure representing the synset to search for a pointer of type *ptr_type* . When a pointer type match is found, the synset pointed to is read is linked onto the *nextss* chain. Levels of the tree generated by a recursive search are linked via the *ptrlist* field structure until **NULL** is found, indicating the top (or bottom) of the tree.

traceptrs_ds is modified into the *traceptrs_ds_new* function. The modified function will restrict the retrieval of synonyms by searching the wordNet with limited number of senses and traverses limited number of levels.

Again, the above are merely illustrative of the many possible applications of the present invention, and it is expected that many more web-based enterprises, as well as other consumer applications (such as intelligent, interactive toys) can utilize the present teachings. Although the present invention has been described in terms of a preferred embodiment, it will be apparent to those skilled in the art that many alterations and modifications may be made to such embodiments without departing from the teachings of the present invention. It will also be apparent to those skilled in the art that many aspects of the present discussion have been simplified to give appropriate weight and focus to the more germane aspects of the present invention. The microcode and software routines executed to effectuate the inventive methods may be embodied in various forms, including in a permanent magnetic media, a non-volatile ROM, a CD-ROM, or any other suitable machine-readable format. Accordingly, it is intended that the all such alterations and modifications be included within the scope and spirit of the invention as defined by the following claims.

What is claimed is: